YAWATT - (yet)
Another web application testing toolkit

HITB 2006
Kuala – Lumpur
Or a “non-monkey” approach to web applications hacking

By fyodor and meder

fygrave@o0o.nu  meder@o0o.nu
“Nope. we are not writing another web scanner!!”
Agenda

- Why hacking web applications
- What scanners do. Why they are useless (or not)
- What else could be done, but isn’t (yet)
- Introduction to YAWATT
  - User-session based approach
  - Distributed
  - Intelligent (or not?)
  - Modular
  - More than “application security scanner” ..
So, why going for the web

- Good Admins learnt to configure their firewalls
- Good Admins disable services they don’t want
- Good Admins even finally know how to use nmap (and even nessus!!)

....

- But Good Admins still need to provide Web
- And they are not programmers
And more...

- The web applications get complex
- New web frameworks make it even more fun (AJAX)
- Due to high demand of web application programmers, many only have “learn {CGI|PHP|perl|ASP|..} in 24 hours” experience
So the Web applications remain the largest hole in one’s network

- The code is bad
  - Q/A not security oriented
  - Must get product to market ASAP
- Firewalls are there – **but** they can’t help
- IDS are there – but they are blind
- Application “firewalls” - stop limited number of web application attacks (basic user input validation), but are useless when it comes to detection of logical vulnerabilities
Scanners evolution - summary

- Nessus et all – don’t see web applications beyond the underlying software configuration.
- Kavado/Webinspect/N-Stalker/Watchfire Appscan – intelligent scanners. Session aware. But closed architecture, “blackbox” (some allow scripted plugins) and costs $$$$$$$$$$$$$$$$$$$$$$$
Why scanners aren't enough

- Single-host based
- Non-extendable, non-correctable.
- Little or no control on “hacking” process execution flow
- Not easily “extend on the fly” with new ‘automation’ methods
- Often primitive, strict signature based logic
What would we like to have

- Maximum automation of web hacking process
- Minimum of code writing.
- Event-driven workflow
- Manual control
More on the wishlist

- Autonomous functionality (you can shutdown, restart, reload modules, provide new data on the fly and so on)
- “Human to machine” knowledge transfer
- Ability to add new ‘hacks’ on the fly
- Deal with uncertainty in “intelligent way”
- Learn from valid user session data
Be able to attack web application from multiple-locations (bypass IP restrictions, improve brute-forcing process)

Be able to automate the testing of application logic bugs

Be able to make intelligent guesses in case of uncertainty
Introducing YAWATT method
YAWATT learns from user sessions

- User sessions – collections of user’s requests and responses (url, name/value pairs, session information and selective HTTP protocol data)

- Classified user session data include semantic classification of URL, parameters, responses and HTTP protocol data (server type, backend system(s) if visible, “unusual” HTTP headers detected and included)
Application content is learnt from user sessions (data feeders: proxies, enumeration tools)
Real-time content analysis with additional verification
Classification

User session data is classified by:

- Semantic and functional classification of URL
- HTTP protocol classifiers (server type, cookies ..)
- Session classifiers
- Input data classification – type, semantics
- Output classification (application error detection, redirects, “bogus’ responses etc)
Classification process as new data arrives into the system
Testing process

- Plugins (tests) could be executed during the collection of user session data if any of user session data triggers certain plugin
- Plugins (tests) are executed on demand, when user session data is completed
YAWATT Intelligence components (components under development)

- Web application components (URL) classification
- Semantic classification for web application input data
- LSI based response analysis (comparison of web content)
  In response analyzers.
- Use of queries to external sources, search engines
- Limited “binary analysis” of downloaded files (decoding pdf, doc, rtf (other formats later))
- Generation of target-specific brute-force dictionaries
Possibility to create new classification rules on the fly (and let the system re-learn from it)

Possibility to ‘reclassify’ application responses

Possibility to add new ‘testing’ plugins and methods on the fly or correct the old ones
How is URL classification used

Vulnerability scenario testing – uses ‘classificators’ subscription mechanism.

- For example: login page tester will need ‘login’, ‘executable’ and ‘session’
Input data classification

- Rule based classification (plugins)
- Input data: page context, parameter name, parameter value, url

Successful?

No, Classify via Bayesian

Yes

Classify

Bayesian Networks based Classification engine

Teach

Classify
Use of classified user session data
Other ideas to work on:

- Detection of “hidden” parameters (“intelligent” fuzzy tests)
- Identification of “hidden” URLs
- Fuzzy recognition of “negative” and ‘positive” responses using LSI
- Detection of application failures, redirects
- Evaluation and priority based execution for plugins
Distributed architecture

- DB Miner
- Spiders
- And other data miners
- Proxies
- PluginManager
- PluginManager
- Data Classifier
- Other agents
- Communication Framework

Hacker (you)
Distributed architecture (another look)

Yet Another Web Application Testing Toolkit (YAWATT)
What distributed approach gives us:

- Heterogeneous environment (different platforms with different software can work together)
- Distributed brute-forcing. Bypassing IP based restrictions, bandwidth limitations
- IDS – more tricks to evade
- Bypass packet filtering restrictions (ability to place agents behind the firewall!)
Communication layer framework in detail:

- Modified version of spread toolkit used as base
  - Robust
  - Reliable message delivery
  - Portable (windows/unix)
  - Available in C/C++ and Java flavours. Bindings exist for Python, Ruby!
  - Spread is used in proof-of concept code and will be ditched in future!
Aside from application vulnerabilities, other things of interest are:

- Email addresses, user ids that could be seen within web content
- Domain names (within web pages, comments, binary files, etc)
- Building ‘target-oriented’ dictionary files (used by brute-force cracking modules)
How the targeted dictionaries for brute-force attacks are generated:

A statistical information extraction method is applied:

- Step 1: Random similarly styled texts in the same language as the target application content, are analyzed and the statistical occurrence of each word is calculated.
- Step 2: Statistical occurrence of each word within the target website is calculated.
- Step 3: The dictionary is produced by selecting those words which probability produced in Step 1 and Step 2 is significantly different.
Add your plugin code on the fly (attack automation plugins via subscription mechanism, classification plugins etc):

- Can’t be simpler:
Look mah, no hands!

No reload is needed, plugins executed next time the new data is processed
DEMO
Trying code

- http://o0o.nu/ - pre-release.
  - You will need:
    - Spread toolkit (www.spread.org)
    - Patched version of Ruby, Spread bindings for ruby.
    - ‘classifier’ package (Bayesian, LSI algorithms), ‘mysql’
    - Burp proxy as data source
    - MYSQL database
Questions and Answers

Sample questions, pick one: ;--------)

- Why another web hacking tool?
- Can you do X too..?
- Can X be integrated too ..?
- This presentation is boring, any excuse ..? 😊
Thanks

- Thanks for your patience
- Send us emails if you try the code
- The code, slides and docs will be available in a while:

  http://o0o.nu/