Notice!

Updated presentation materials are available online at:

http://www.wiretrip.net/rfp/blackhat-asia/

Assessing the web

A look at the tools used to secure online applications

Rain Forest Puppy
rfp@wiretrip.net

Why target the web?

- It's everywhere! Mobile phones, cars, watches, toasters…
- Safe bet the protocol will not become obsolete anytime soon
- New technology is being implemented/retrofitted on top (e.g. SOAP, WebDAV)
- Protocol fundamentally not suited to do a majority of what it's doing today ('kludge job' galore)

Problem areas in HTTP

- The protocol is extremely simple; therefore, it is easy to (mis)code your own HTTP server
- Lack of experience coding public-service, multi-user applications
- Multitude of involved technologies
- Stateless nature…

The ‘stateless’ problem

- The HTTP protocol doesn’t include the mechanisms needed to have data persistence between requests
- People are left to implement there own ‘attempt’ at data persistence (retrofit state onto the stateless protocol)
- Same typical problems reoccur due to everyone ‘reinventing the wheel’
- Extremely difficult, if not impossible, to invent tools that can automatically assess such custom implementations

Tools as they exist today

Vulnerability scanners

- Look for a known list of vulnerable applications or technologies
- Do not (can not) engage/scan custom applications and configurations
- General vulnerability scanners: ISS, Cybercop, Nessus
- Web-specific vulnerability scanners: WebInspect, whisker

‘Proxy monitors’

- HTTP proxy which monitors traffic, looking for web vulnerabilities as they pass
- Can analyze custom applications with the help of a user
- Examples: AppScan, RFProxy

Quick spotlight

- Sanctium AppScan – proxy that passively monitors for potential problems
- Philip Stoev's ELZA – scripting tool used to interact with web applications
- Spi Dynamic's WebInspect – vulnerability scanner that crawls website
- eEye Retina – CHAM can be used to methodically check for buffer overflows
- General vulnerability scanners – Nessus, ISS, Cybercop, HackerShield, etc

The coming of whisker

whisker 1.x

- Released October 20, 1999
- ‘Competition’: VoidEye, cgichk, billions of home-coded scanners
- Coded out of shear annoyance by currently-available scanners

whisker 1.x design goals

- Scriptable: easily updated by just about anyone
- Intelligent: conditional scanning, reduction of false positives, directory checking
- Flexible: easily adapted to custom configurations
- Bonus features: IDS evasion, virtual hosts, authentication brute forcing

whisker 1.x fallout

- Very few people actually wrote new scripts; those capable of writing scripts seemed capable of writing full code
- Still requires some forethought and tweaking to fully assess custom configurations
- Code design could be improved; some functionality was still lacking
- Limited to ‘file-existence’ scanning

Back to the drawing board

Various personal observations

- Most people shifted to a new scanner when old one became outdated
- People reusing whisker's sendraw() HTTP code
- To be viable, scripting support had to be full-featured
- People tend to (poorly) recode common functionality in web exploits and demonstration code

So how about a new approach…

- A well-documented and full-featured library of components useful to assessing web applications
- Replaces the limited-scripting model with the full capabilities of a real programming language
- Lends better to the custom nature of web applications

And thus whisker 2.x…..a.k.a. ‘libwhisker’

- All whisker functionality has been modularized into various Perl functions
- ‘Scripts’ now have the full processing power of Perl to use for logic evaluations
- Perl is relatively cross-platform portable
- Since those who coded their own whisker 1.x scan scripts seemed capable of coding Perl, this isn’t a large loss in my eyes

“I can’t code, what good does an API do me?”

Directly? Not much good. However, there are three important points that do affect you:

- Modular nature of whisker will hopefully spark new interest in development and contribution (I can’t continue to be a one-puppy-show)
- If people use libwhisker as a basis for their code, you indirectly receive the stability in other programs
- A script that demonstrates the API will be included; however, this script will serve all the functions of the normal whisker scanner

Peek into libwhisker

HTTP module

- The core of whisker, passes a request to a server and receives the response
- Capable of handling HTTP 0.9, 1.0, and 1.1 connections
- HTTP 1.1 keep-alive/connection reuse
- Receive chunked encoding
- Full integrated HTTP proxy and virtual host support
- Complete cross-platform timeout support
- Transparent data handling (PUT, POST, etc)
- Handles HTTP ‘100 Continue’ responses

HTTP module cont.

- All aspects are completely customizable for ultimate control
- Controlled by a single structure (Perl hash)

How easy is it to make a simple web request?

```
use libwhisker;
my %in, %out;
whttp::request_init(\%in);
$in{‘whisker’}->{‘uri’}=‘/some/webpage.htm’;
$in{‘whisker’}->{‘host’}=‘www.wiretrip.net’;
whttp::do_request(\%in,\%out);
print $out{‘whisker’}->{‘http_resp’};
```

Other modules

- Auth: basic, MD5, and NTLM authentication support routines
- BruteURL: brute forcing support
- HTML: HTML parsing routines
- Crawl: site/link crawling
- Encode: various encoding routines (hex, Unicode, etc)
- DAV: WebDAV request wrappers
- FrontPage: MS FrontPage client emulation
- IDS: anti-IDS routines (6 new methods for a total of 15 anti-IDS tactics)

“Sounds good, when can I play?”

Today! The preview release of whisker 2.0 (whisker pr2) can be downloaded from:
http://www.wiretrip.net/rfp/blackhat-asia/

The preview release is meant to introduce the concept of libwhisker, and hopefully solicit developers willing to work on and contribute to the project further. Not all modules and/or functionality will be in the preview release.

“How about a demonstration of libwhisker?”

I suppose I could, but demonstrating an API can be quite boring (particularly to non-coders). The preview release does not include any demonstration scripts—it is just the API. So for the non-coders in the crowd, you may feel left out.

However, I don’t want anyone to feel left out, so….

Let’s play: using RFProxy

First off, what is RFProxy?

RFProxy is a web assessment tool to be used by an individual to help identify and exploit vulnerabilities in online applications.

It accomplishes this task by acting as an HTTP proxy, essentially ‘extending’ the features of the user’s normal browser to be more suited for security testing.

RFProxy vs. other tools

- There **are** other proxy-based tools currently available to help assess web applications (in particular, Sanctium's AppScan)

- AppScan passively monitors passing HTTP traffic, looking for vulnerabilies

- RFProxy, on the other hand, actively interacts with the HTTP traffic (e.g. rewriting the HTML)

What can RFProxy do?

- Hidden form fields become visible (and editable)
- Radio, checkbox, and select fields can have arbitrary values
- Maxlength limitations are removed
- Javascript value checking is removed
- Arbitrary headers can be added, deleted, or modified (such as Referer)
- Cookies can be added, deleted or modified
- Requests can be captured, modified, and replayed
- Plus tons of support tools are available via web interface

RFProxy uses

- Testing dependencies on client-imposed limitations
  (such as form field values and Javascript checks)
- Evaluating state tracking and authentication mechanisms
- Looking for potential ways to perform abusive or
  fraudulent online transactions

Eyecandy: RFProxy demo

RFProxy development

- RFProxy will eventually be built on libwhisker
- More passive-monitoring intelligence is planned for future releases
- RFProxy will take advantage of the same pool of resources as libwhisker

Questions

Thanks

Rain Forest Puppy
rfp@wiretrip.net