

Ferret: A Host Vulnerability Checking Tool

Anil Sharma

Jason R. Martin

Nitin Anand

Michel Cukier

William H. Sanders

Outline

- Introduction
 - Definitions
 - Previous Tools
 - Introducing Ferret
- Ferret Design Goals
- Detailed Design
 - Architecture Overview
 - Ferret Core
 - Vulnerability Checking Plug-ins
 - Output Plug-ins
- Experimental Results
- Future Work and Conclusions
- Thanks

Definitions

- *Attack* – a malicious act that attempts to exploit a weakness in the system
- *Vulnerability* – an accidental fault, or a malicious or non-malicious intentional fault
- *Intrusion* – an attack that has been at least partially successful
- An attack is thus an intrusion attempt

Previous Tools

- *COPS* (Computerized Oracle and Password System) – Created in 1990 by Daniel Farmer from CERT at Carnegie Mellon and Eugene H. Spafford at Purdue. Last updated in 1999. <http://dan.drydog.com/cops/>
- *Tiger/TARA* – Created by the CIS Network group of Texas A&M, previously updated in 1994. Recently new development has started (late 2002). <http://www.tigersecurity.org/>

Introducing Ferret

fer-ret *n.*

1. A partially domesticated usually albino European polecat.
2. An active and persistent searcher.

Ferret was created to fill the void left by the previous scanning tools

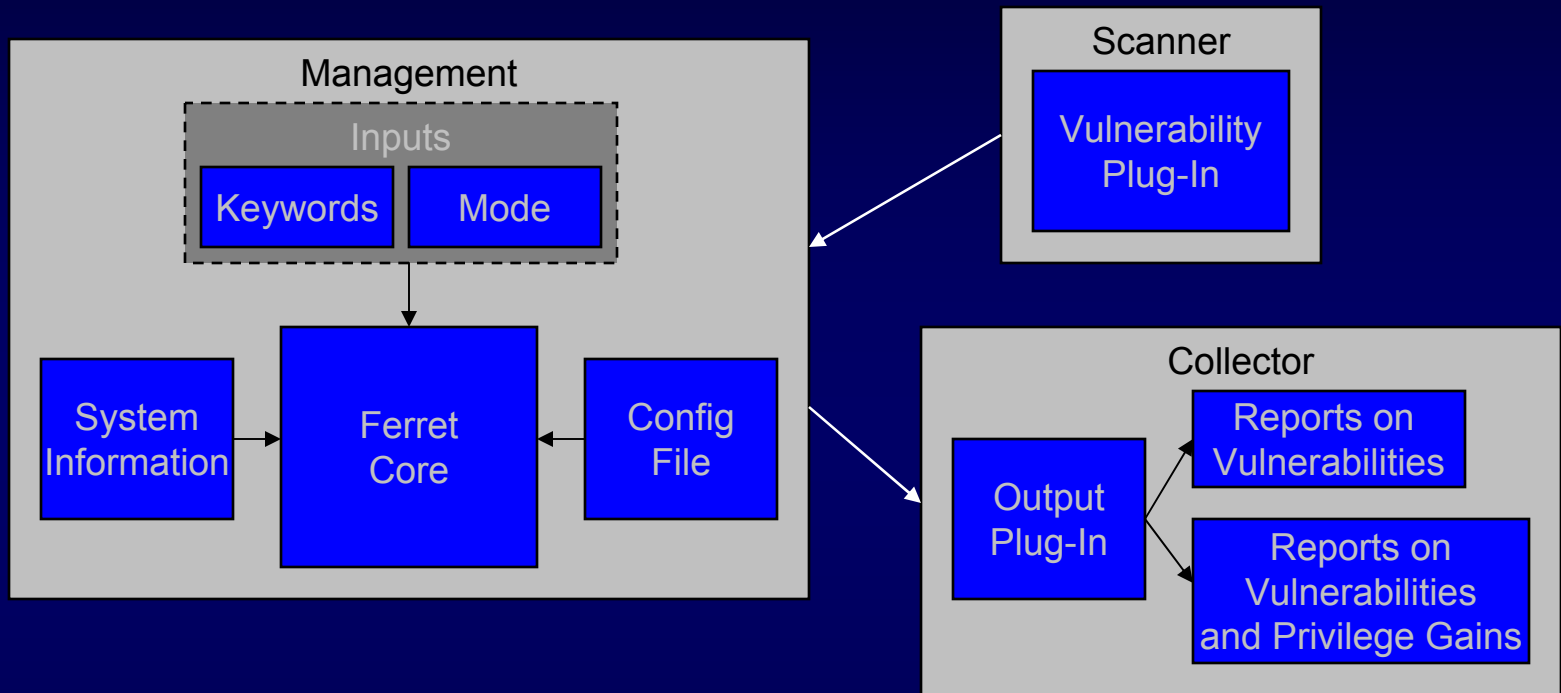
Ferret Design Goals

- Modular vulnerability checking – should be easy to add new vulnerabilities as they are discovered
- Vulnerability checks as specific as possible – each vulnerability check should only check one specific vulnerability
- Open source, with active development community – proven method of keeping current with vulnerabilities (reference Nessus, Snort, etc.)
- Simple design to increase performance

Ferret Design Goals (cont.)

- Platform-independent – must include support for platform dependent plug-ins, but tool itself should be independent
- Keyword-grouping of vulnerability plug-ins – allows plug-ins to be run based on context (i.e. plug-ins checking /etc/passwd vulnerabilities)
- Modular Output – generation of reports based on data collected should be up to the user running the tool
- Built-in support for privilege escalation analysis (privilege graphs)

Architecture Overview



Three main architecture areas:

- Management (*Ferret Core*)
- Scanning agent (*Vulnerability Plug-ins*)
- Collecting agent (*Output Plug-ins*)

Ferret Core

- Designed and implemented in Perl
 - Designed for text file processing
 - Popular and widely available
- Selects and runs individual vulnerability plug-ins based on keywords specified
 - Allows vulnerabilities to be loosely classified and checked together
- Feeds output of individual vulnerability plug-ins to selected output plug-in to produce scan report
 - Can generate reports in different formats an output plug-in is written for
 - Examples include plain text, html tagged, or output to a MySQL database

Ferret Core Procedure

1. Interpret command line, read in config file
2. Determine OS and version
3. Determine what plug-ins are available
4. Query plug-ins for meta-data
5. Build hash table of plug-in keywords
6. Determine which plug-ins to run based on keywords (default keyword of *all*)
7. Run plug-ins
8. Divide plug-ins into two categories: those that found vulnerabilities and those that did not
9. Pipes output through the output plug-ins based on chosen mode and output format

Vulnerability Plug-ins

- Designed to scan for one specific vulnerability
- Interact with Ferret Core using command line options
- Can either check for vulnerability themselves or provide a wrapper to another tool (such as John the Ripper, a password cracking tool)
- Can be implemented any way desired, as long as they accept the above command line options

Vulnerability Plug-ins (cont.)

- Currently about 80 plug-ins implemented
- Can be loosely divided into nine groups
 - Critical system files/directories are owned by root
 - Critical system files/directories are world writable
 - Paths and filenames in root start-up files world writable
 - Umask settings in start-up files
 - Configuration file permissions in user home directories
 - SUID file permission checks
 - Password file configuration issues
 - Exported filesystem and mounted filesystem configurations
 - .rhost file settings

Output Plug-ins

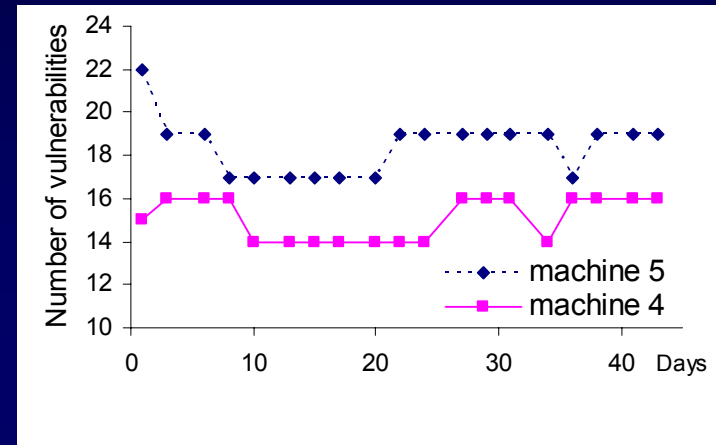
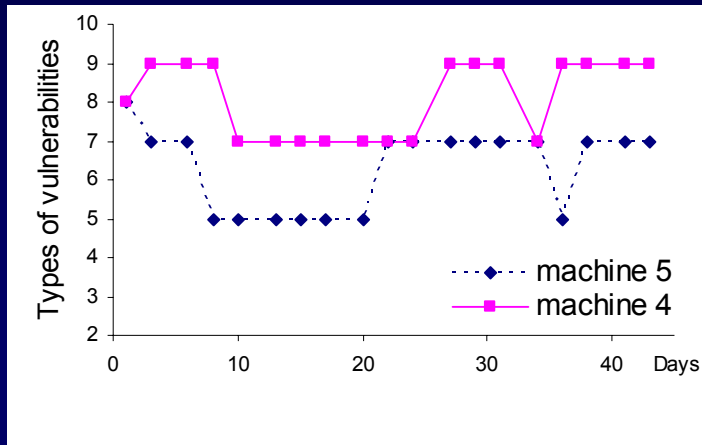
- Two different modes currently implemented
 - Vulnerability Information
 - Output includes name of plug-in, short description of the plug-in, result of the scan, and additional output generated by the plug-in
 - Vulnerability and Exploitation Information
 - Produces additional output useful for generating privilege graphs
 - Each node represents a set of privileges assigned to a user or set of users
 - Any arc that connects two nodes is representative of a system vulnerability

Experimental Results

- Ferret installed and run on six machines at UIUC
- Data collected three times per week for 1.5 months
- Three server-class machines and three workstations

Machine	Operating System	OS Version	Machine Function	Types of Vulnerabilities	Number of Vulnerabilities	Execution Time (sec)
1	Linux	RH7.2 + Updates	Workstation	22	102	146
2	Sun Solaris	5.8 (Solaris 8)	Workstation	2	4	29
3	Sun Solaris	5.8 (Solaris 8)	Server (NIS, NFS, RPC)	12	14	234
4	Sun Solaris	5.8 (Solaris 8)	Server (NIS, NFS, RPC)	8	19	180
5	Sun Solaris	5.7 (Solaris 7)	Workstation	6	15	25
6	HP-UX	B.10.20	Server (NIS, NFS, DNS, DHCP, Mail)	35	303	111

Experimental Results (cont.)



- Graphs plotted for machines 4 and 5
- Time span too short to draw conclusions
- Most changes took place in user home directories

Future Work and Conclusion

- Useful tool that automates significant portions of a system administrator's security duties
- Needs active security community involvement
- Expand beyond Linux, Solaris, and HP-UX platforms
- Develop more vulnerability and output plug-ins
- Collect data on more networks over longer periods of time
- Integrate results with broader quantitative security evaluation efforts

Thanks

The Coordinated Science Lab system
administrators that made testing possible:

Michael Chan

Loren Heal

Gabriel Lopez-Walle

Additional testing and programming suggestions
provided by Michael Chan, Tod Courtney, and
Loren Heal