NETWORK MONITORING AND DECEPTIVE DEFENSES

Michael Collins, RedJack	mpcollins@redjack.com
Brian Satira, Noblis	Brian.satira@noblis.org

INTRODUCTION

INTRODUCTION

- Statement of problem
- Experimental System
  - Model of Attack
  - Defensive System
  - Results
- Future Work
  - Game Based Evaluation
  - Implementing Defense

THE PROBLEM OF PASSIVITY

- IDS is historically *passive*
  - Collects and models data representing 'normal' traffic
- Normal has been hard to define
  - Paxson & Floyd, "Why We Don't Know How To Simulate The Internet"
  - Gates & Taylor, "Challenging the Anomaly Detection Paradigm"

ALTERNATIVE APPROACH

- *Engineer the network to be detection-friendly*
  - Provide defenders with *more options* to frustrate the attacker
- We leverage deceptive techniques
  - Artificially inflate files to make them hard to exfiltrate without being noticed
  - Add in honeyfiles that look like target files to force the defender to make a choice

WHY DECEPTION?

- Spafford, “Planning and Integrating Deception Into Computer Security Defenses”
- Argues that we’ve been caught in a misinterpretation of Kerckhoffs’

The misinterpretation has led many security practitioners to believe that any “obscurity” is ineffective, which is not the case. Hiding a system from an attacker or having a secret password does increase the work factor for the attacker— until the deception is detected and defeated. So long as the security does not materially depend on the ”obscurity,” the addition of misdirection and deceit provides a defensive advantage.

EXPERIMENTAL SYSTEM

ATTACK MODEL: EXFILTRATION

- Attack penetrates site, copies content and extracts data
- Historical examples
  - Snowden (2014)
  - Manning
  - HBGary

DETECTING EXFILTRATION

Measure	Counter
Watch for sensitive terms crossing network boundary	Encrypt transferred data
Look for malware signatures	Randomize malware, use common tools that aren't malware
Look for spikes in traffic at specific times	Spread out exfiltration over an extended period
Look for spikes in traffic to specific hosts	Spread out exfiltration to multiple targets
Aggressively monitor transfer channels (HTTP)	Move exfiltration over to less obvious protocols (DNS)

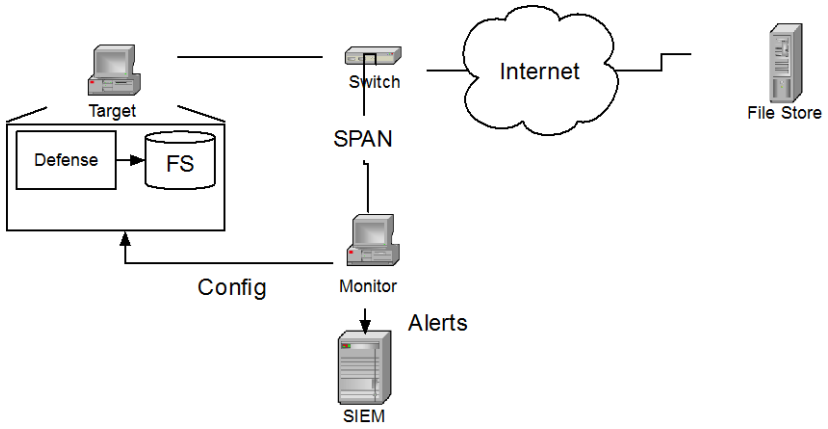
THE POINT

- DLP (data loss prevention) is a process, we have to constantly add new defenses to deal with attacker responses
- The goal of this work is to give defenders *new tools,* outside of the passive model
- We can, however, approach the problem strategically
  - We want to back the attacker into a corner

THE CORNER

- Make a class of automated exfiltration *harder* for the attacker
- Key observation: attackers search *shallowly*
  - Malware hunts for keywords
  - Insiders just copy everything
- We win when:
  - The attacker has to manually examine documents on site (forcing him to reveal himself)
  - The attacker picks bad data (failing to exfiltrate)
  - The attacker reveals himself when revealing data

A DECEPTION CONTROL LOOP



- We add a monitoring system that examines outbound traffic characteristics
- We monitor the traffic to determine a reasonable maximal file size

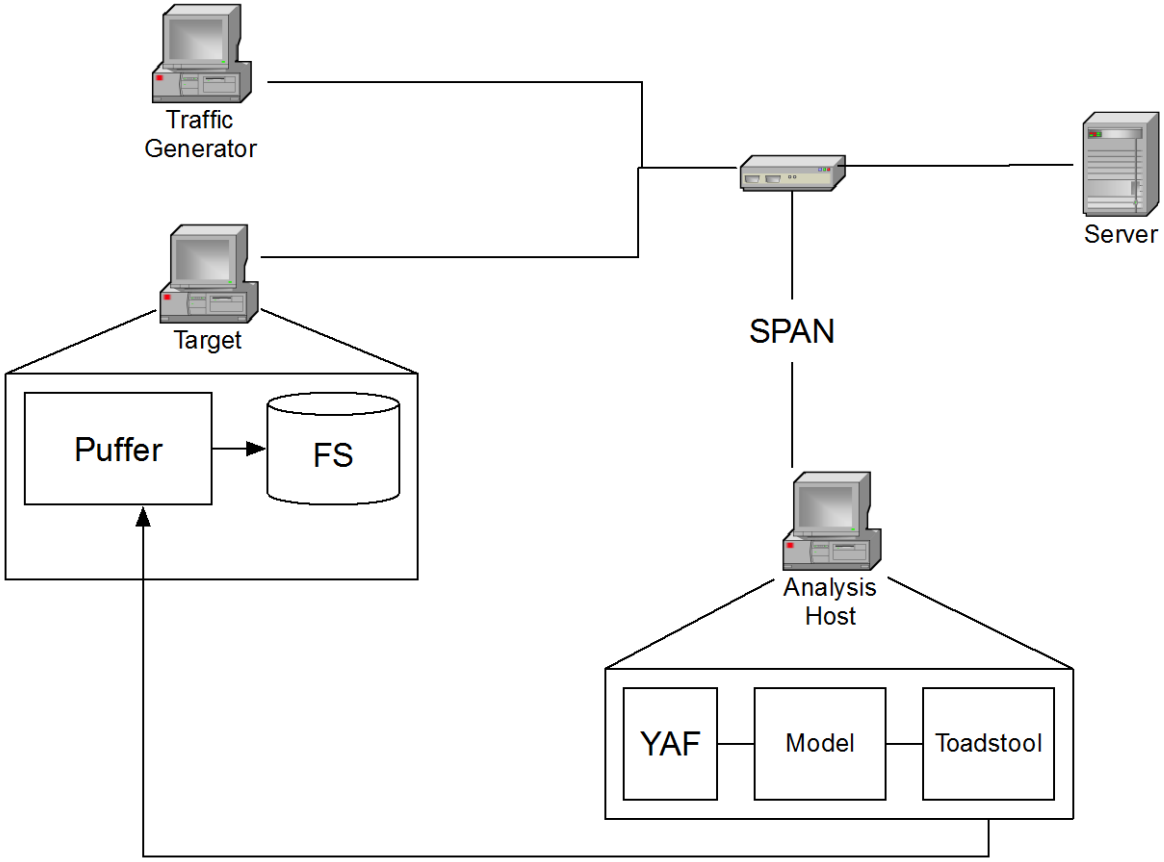
POC SYSTEM GOALS

1. Demonstrate feedback loop
   - Build a system that monitors traffic…
   - …identifies file inflation threshold…
   - …updates hosts appropriately
2. Determine whether file inflation is practical
   - Are the file sizes feasible?
3. Complete end-to-end system
   - Monitor, configure, alert

DEFENSIVE COMPONENTS

- Traffic monitoring system (YAF + Toadstool)
  - Monitors outgoing traffic
  - Develops model of outgoing behavior
  - Estimates file size required to raise an alert
- File inflator (Puffer)
  - Fills office files with extra data
- Deceptive text generator (SciGen)
  - Creates bogus text

TESTBED SCHEMATIC



FEEDBACK LOOP

- Monitoring system implemented as a cron job, collects data from YAF and builds a timing model
  - Model is a simple empirical CDF
- Based on the model, we calculate a threshold on our FPR
  - FPR translated to “alerts handled per day”, e.g., 95% @ 900 second intervals ~ 4 apd
- Threshold is then fed into a specialized client (Toadstool) that feeds the threshold filesize to Puffer

FEEDBACK LOOP (II) - PUFFER

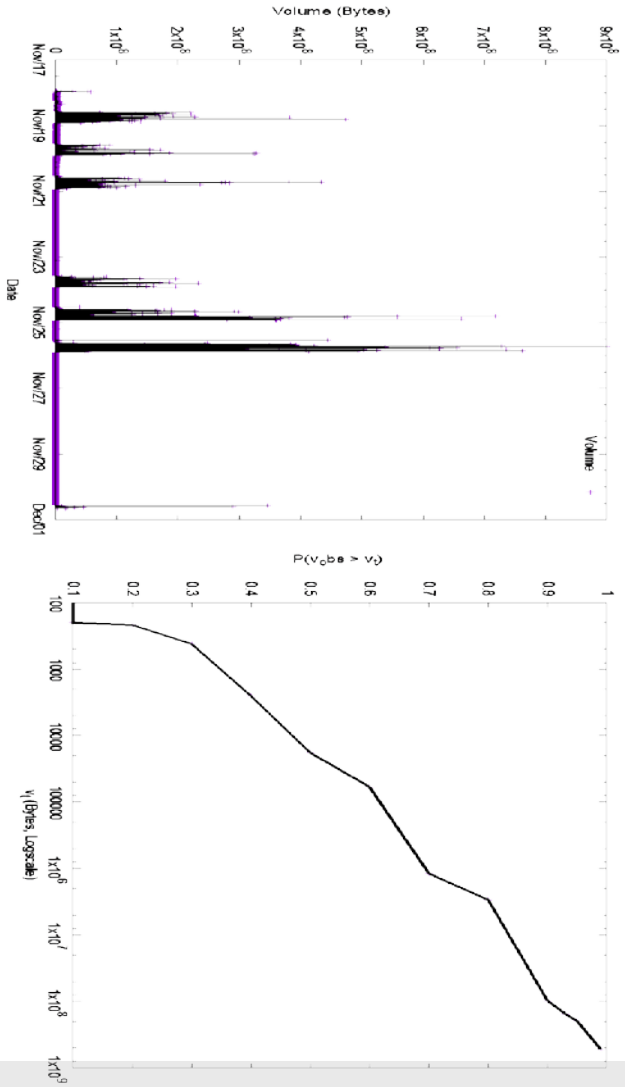
- Puffer is a command line application that inflates OOXML files
  - MSWord files are zip archives
  - We add another file containing bogus information

FEEDBACK LOOP RESULTS

- Measurement → Toadstool → Puffer is feasible
- Windows/Office is the weak link in the chain
- Most notable problem – files inflated > 512 MB cause problems
  - Max text size is 32 MB, but we can add images
  - Files > 512 MB may be autocut down to < 512
  - Windows raises error messages
- 512 MB limit is the most significant problem, since it governs how big a file we can create

FILE SIZE INFLATION

- Traffic volumes measured from small (20-100 host network) over 2 weeks
- Results are spiky, there is a long tail
  - 90% < 98 MB/15min
  - 95% < 196 MB/15 min
  - 99% < 525 MB/15 min



THE FEEDBACK LOOP CAN WORK

- If we set inflation to 200 MB, we pay ~4 alerts/day
- At 1 alert/day, we're going to reach the limits of Word *so far*

FUTURE WORK

GOAL 1: BETTER DECEPTION

Measure	Counter
Use compression to reduce overhead	Generate high entropy bogus text to reduce the compressibility of data
Publicly dump data	Incorporate steganographic information containing log data (IP addresses contacted, time of extraction)
Copy multiple files	Add multiple false files generated with keywords

GOAL 2: UNIFIED MODEL

- Build a Stackelberg game to model attacker response
  - Attacker's reward function: value of assets extracted vs. data examined
  - The more the defender forces the attacker to invest time and attention on files, the more likely the defender can find him

GOAL 3: DEFENSE AS A PROCESS

- Evaluate feasibility over different architectures and configurations
- The goal is to “add defender tools”
  - Tools may not be applicable everywhere
  - Figure out limits

CONCLUSIONS

- Deceptive defense, combined with network monitoring, opens up the potential for new defensive methods
- We can engineer networks to be more “detection friendly”, combining multiple systems at the host and the network together
- Successful integration will provide tools, but those tools may not be universally applicable
  - Pre-testing and gaming become critical elements