LA-UR- 11-04114

Approved for public release; distribution is unlimited.

Title: Detecting Code Injection Attacks in Internet Explorer

Author(s): Blake Anderson
Daniel Quist
Terran Lane

Intended for: IEEE Computer Software and Application Conference 2011 -- presentation

Los Alamos
NATIONAL LABORATORY
EST. 1943

Detecting Code Injection Attacks in Internet Explorer

Blake Anderson
Los Alamos National Lab
banderson@lanl.gov

Daniel Quist
Los Alamos National Lab
dquist@lanl.gov

Terran Lane
University of New Mexico
terran@cs.unm.edu

Abstract

Code injection vulnerabilities are a major threat to Internet security. The ability for a malicious website to install malware on a host using these vulnerabilities without its knowledge is particularly menacing. In this paper, we approach this problem from a new perspective by constructing a Markov chain graph from the system calls Internet Explorer executes and then modeling this graph over time. We apply a Gaussian process change-point algorithm to detect code injection attacks. To show the efficacy of this approach, we collect a novel dataset of system call traces of 6 code injection attacks using 3 distinct exploits against the Internet Explorer browser. Our algorithm was able to detect all of the code injection attacks with a limited number of false positives.

Detecting Code Injection Attacks in Internet Explorer

Blake Anderson, Daniel Quist, Terran Lane

Introduction

- **The Internet is a dangerous place**
- **Malicious downloads, Drive-by downloads**
- **Collect system call traces of Internet Explorer**
- **Use change-point detection algorithms to find changes in the application's behavior over time**
- **Real-world example of concept drift (with ground truth)**

Code Injection Attacks

- **Just visiting a website can lead to infection**
- **Javascript is used to host the shellcode**
- **Exploits typically use vulnerabilities in Internet Explorer's associated plug-ins**
- **The plug-ins are called within javascript, and the exploit is initiated**

Exploits

- **Aurora**
  - Uses an invalid pointer reference within Internet Explorer
  - Creates a backdoor connection to a command and control server
- **Dllloader**
  - IE calls DLLs with relative paths
  - Icon is presented to the user, user downloads dll, IE then calls the dll
- **Unsafe Scripting**
  - Takes advantage of a buffer overflow exploit in the ActiveX controls of IE

Data Collection

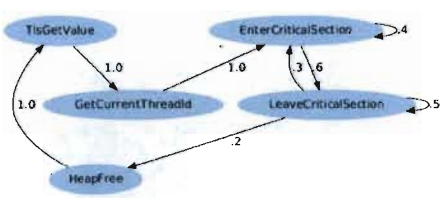
- **Xen Virtual Machine**
  - Provides a safe environment to perform exploits
- **Metasploit**
  - Many exploits are already written
  - Easy to use interface
- **StraceNT**
  - Collects the system calls made by the process

Data Collection: Part II

- **Collect normal IE usage for 10 or 15 minutes**
- **Initiate exploits, then let IE run for 5 or 10 minutes (20 minutes total)**
- **This process results in files with ~1-1.4 million system calls**
- **Each time bin consists of 10,000 system calls**

Data Representation

- **Markov chain**
  - Vertices are the system calls performed by the process
  - Transition probabilities (edges) are estimated by the traces



Change-Point Detection Algorithm

- **Goal: find** $p(r_t|x_{1:t})$

$$p(r_t|x_{1:t}) = \frac{p(r_t, x_{1:t})}{\sum_{r_t} p(r_t, x_{1:t})}$$

- **Use recursive message passing scheme to find** $p(r_t, x_{1:t})$

$$p(r_t, x_{1:t}) = \sum_{r_{t-1}} p(x_t|x_{t-1})\, p(r_{t-1}, x_{1:t-1})$$

- $p(x_t|x_{t-1})$ **Is given by a Gaussian process time-series model**

Change-Point Detection Algorithm: Part II

- $p(x_t | x_{t-1}) = N(m_t, v_t)$ where

$$m_t = K_s^T (K + \sigma^2 I)^{-1} \mathbf{x}$$

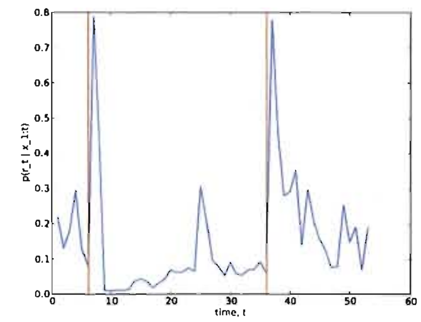
$$v_t = K_{ss} - K_s^T (K + \sigma^2 I)^{-1} K_s$$

- We use the Gaussian kernel

$$K(\mathbf{x}, \mathbf{y}) = \sigma^2 e^{-\frac{1}{2\lambda^2} \sum_{i,j} (x_{ij} - y_{ij})^2}$$

Synthetic Results

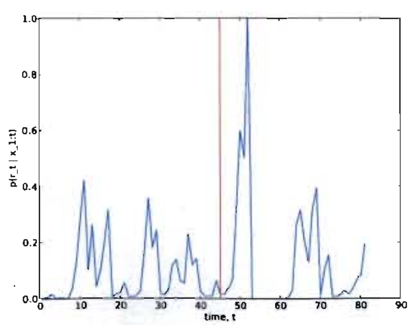
- Benign and malicious instruction traces
- 5,000 instructions per bin
- Inserted malicious instruction bins in two places within the benign trace



Los Alamos NATIONAL LABORATORY
Operated by Los Alamos National Security, LLC for the U.S. Department of Energy's NNSA
NNSA

Real-World Results: Aurora

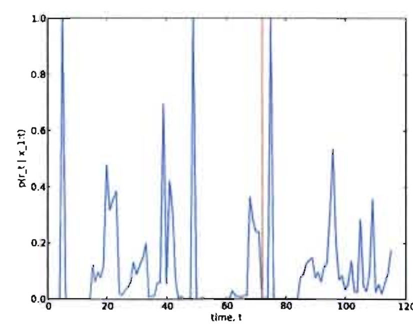
- Bins of 10,000 system calls
- Ran for 10 minutes, exploit, ran another 10 minutes



Los Alamos NATIONAL LABORATORY
UNCLASSIFIED
Operated by Los Alamos National Security, LLC for the U.S. Department of Energy's NNSA
NNSA

Real-World Results: dllloader

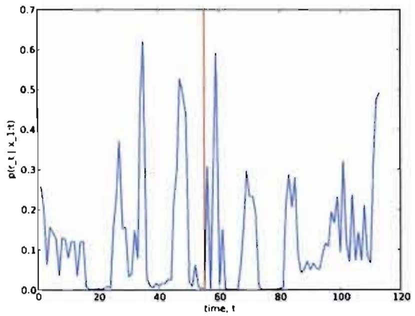
- Bins of 10,000 system calls
- Ran for 15 minutes, exploit, ran another 5 minutes



Los Alamos NATIONAL LABORATORY
UNCLASSIFIED
Operated by Los Alamos National Security, LLC for the U.S. Department of Energy's NNSA
NNSA

Real-World Results: unsafescripting

- **Bins of 10,000 system calls**
- **Ran for 10 minutes, exploit, ran another 10 minutes**



Conclusions

- **We are able to detect when attacks happen**
- **The number of false positives will be prohibitive in a production setting**
- **Ideas to lower false positive rate:**
  - Use instruction traces instead of system call traces
  - More advanced kernels in the GPTS model
  - Using a simpler change-point model