CO 445H

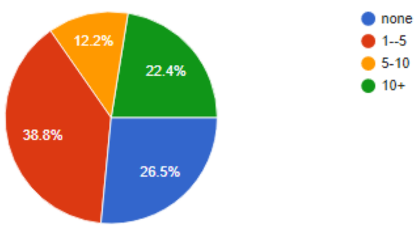
SECURE DESIGN PRINCIPLES
JAVA SECURITY
ROP AND ADVANCED EXPLOITS

Dr. Ben Livshits

Some Survey Responses

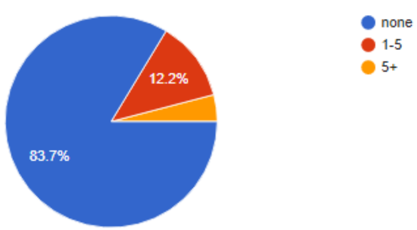
I have read the following number of research papers in computer science

49 responses



I have written the following number of research papers

49 responses



PAPER SUMMARIES:

> As part of the course, you'll be expected to submit three paper summaries for the papers -- you'll get to pick which papers you want to write the reviews for. This should be individual work. Check the calendar for the paper review submission date.

Sunsetting Google+

SAFETY AND SECURITY

Project Strobe: Protecting your data, improving our third-party APIs, and sunsetting consumer Google+

Ben Smith
Google Fellow and Vice President of Engineering

Published Oct 8, 2018

Many third-party apps, services and websites build on top of our various services to improve everyone's phones, working life, and online experience. We strongly support this active ecosystem. But increasingly, its success depends on users knowing that their data is secure, and on developers having clear rules of the road.

Over the years we've continually strengthened our controls and policies in response to regular internal reviews, user feedback and evolving expectations about data privacy and security.

At the beginning of this year, we started an effort called Project Strobe—a root-and-branch review of third-party developer access to Google account and Android device data and of our philosophy around apps' data access. This project looked at the operation of our privacy controls, platforms where users were not engaging with our APIs because of concerns around data privacy, areas where developers may have been granted overly broad access, and other areas in which our policies should be tightened.

We're announcing the first four findings and actions from this review today.

Finding 1: There are significant challenges in creating and maintaining a successful Google+ product that meets consumers' expectations.

Action 1: We are shutting down Google+ for consumers.

https://www.blog.google/technology/safety-security/project-strobe/

Details

Our review showed that our Google+ APIs, and the associated controls for consumers, are challenging to develop and maintain. Underlining this, as part of our Project Strobe audit, we discovered a bug in one of the Google+ People APIs:

- Users can grant access to their Profile data, and the public Profile information of their friends, to Google+ apps, via the API.
- The bug meant that apps also had access to Profile fields that were shared with the user, but not marked as public.
- This data is limited to static, optional Google+ Profile fields including name, email address, occupation, gender and age. (See the full list on our developer site.) It **does not** include **any other data** you may have posted or connected to Google+ or any other service, like Google+ posts, messages, Google account data, phone numbers or G Suite content.
- We discovered and immediately patched this bug in March 2018. We believe it occurred after launch as a result of the API's interaction with a subsequent Google+ code change.
- We made Google+ with privacy in mind and therefore keep this API's log data for only two weeks. That means we cannot confirm which users were impacted by this bug. However, we ran a detailed analysis over the two weeks prior to patching the bug, and from that analysis, the Profiles of up to 500,000 Google+ accounts were potentially affected. Our analysis showed that up to 438 applications may have used this API.
- We found **no evidence** that any developer was aware of this bug, or abusing the API, and we found **no evidence** that any Profile data was misused.

Notifications

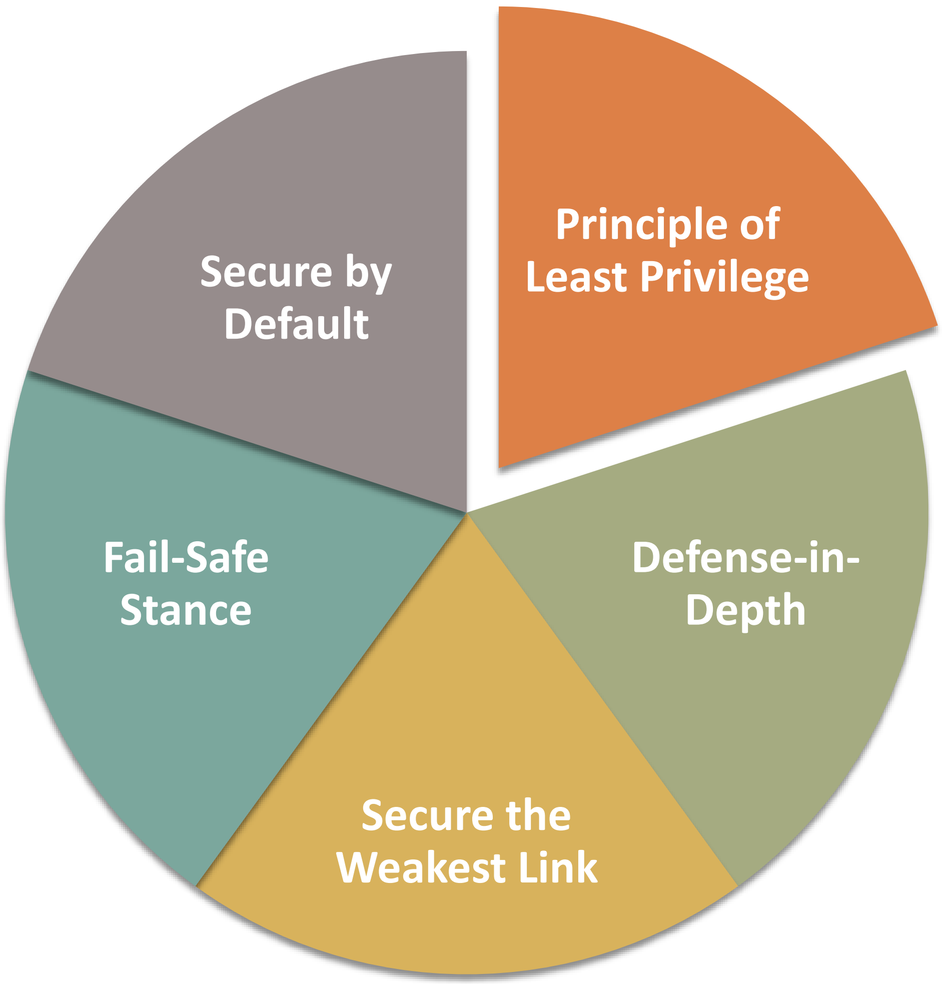
Every year, we send millions of notifications to users about privacy and security bugs and issues. Whenever user data may have been affected, we go beyond our legal requirements and apply several criteria focused on our users in determining whether to provide notice.

Our Privacy & Data Protection Office reviewed this issue, looking at the type of data involved, whether we could accurately identify the users to inform, whether there was any evidence of misuse, and whether there were any actions a developer or user could take in response. None of these thresholds were met in this instance.

The review did highlight the significant challenges in creating and maintaining a successful Google+ that meets consumers' expectations. Given these challenges and the very low usage of the consumer version of Google+, we decided to sunset the consumer version of Google+.

To give people a full opportunity to transition, we will implement this wind-down over a 10-month period, slated for completion by the end of next August. Over the coming months, we will provide consumers with additional information, including ways they can download and migrate their data.

Some of the Common Principles



1) Least privilege

Least Privilege for qmail

- In March 1997, I took the unusual step of publicly offering $500 to the first person to publish a verifiable security hole in the latest version of qmail: for example, a way for a user to exploit qmail to take over another account.
- My offer still stands. Nobody has found any security holes in qmail. I hereby increase the offer to $1000.
- How can we make progress?
  - Answer 1: eliminating bugs
  - Answer 2: eliminating code
  - Answer 3: eliminating trusted code

Some thoughts on security after ten years of qmail 1.0

Daniel J. Bernstein
Department of Mathematics, Statistics, and Computer Science (M/C 249)
University of Illinois at Chicago, Chicago, IL 60607–7045, USA
djb@cr.yp.to

ABSTRACT

The qmail software package is a widely used Internet-mail transfer agent that has been covered by a security guarantee since 1997. In this paper, the qmail author reviews the history and security-relevant architecture of qmail; articulates partitioning standards that qmail fails to meet; analyzes the engineering that has allowed qmail to survive this failure; and draws various conclusions regarding the future of secure programming.

Categories and Subject Descriptors

D.2.11 [**Software Engineering**]: Software Architectures—*bug elimination, code elimination*; D.4.6 [**Operating Systems**]: Security and Protection; H.4.3 [**Information Systems Applications**]: Communications Applications—*electronic mail*

General Terms

Security

Keywords

Eliminating bugs, eliminating code, eliminating trusted code

1. INTRODUCTION

1.1 The bug-of-the-month club

Every Internet service provider runs an MTA (a "Mail Transfer Agent"). The MTA receives mail from local users; delivers that mail to other sites through SMTP (the Internet's "Simple Mail Transfer Protocol"); receives mail from other sites through SMTP; and delivers mail to local users.

I started writing an MTA, qmail, in 1995, because I was sick of the security holes in Eric Allman's "Sendmail" software. Sendmail was by far the most popular MTA on the Internet at the time; see, e.g., [4]. Here's what I wrote in the qmail documentation in December 1995:

> Every few months CERT announces Yet Another Security Hole In Sendmail—something that lets local or even remote users take complete control of the machine. I'm sure there are many more holes waiting to be discovered; Sendmail's design means that any minor bug in 41000 lines of code is a major security risk. Other popular mailers, such as Smail, and even mailing-list managers, such as Majordomo, seem just as bad.

Fourteen Sendmail security holes were announced in 1996 and 1997. I stopped counting after that, and eventually I stopped paying attention. Searches indicate that Sendmail's most recent emergency security release was version 8.13.6 in March 2006; see [10] ("remote, unauthenticated attacker could execute arbitrary code with the privileges of the Sendmail process").

After more than twenty years of Sendmail releases known to be remotely exploitable, is anyone willing to bet that the latest Sendmail releases are not remotely exploitable? The announcement rate of Sendmail security holes has slowed, but this fact doesn't help the administrators whose systems have been broken into through Sendmail security holes.

1.2 The qmail release

I started serious code-writing for qmail in December 1995. I had just finished teaching a course on algebraic number theory and found myself with some spare time. The final kick to get something done was a promise I had made to a colleague: namely, that I would run a large mailing list for him. Sendmail didn't offer me the easy list administration that I wanted, and it seemed to take forever to deliver a message (serially!) to a long list of recipients, never mind Sendmail's reliability problems and security problems.

The 7 December 1995 version of qmail had 14903 words of code, as measured by

```
cat *.c *.h | cpp -fpreprocessed \
| sed 's/[_a-zA-Z0-9][_a-zA-Z0-9]*/x/g' \
| tr -d ' \012' | wc -c
```

(Other complexity metrics paint similar pictures.) The 21 December 1995 version had 36062 words. The 21 January 1996 version, qmail 0.70, had 74745 words. After watching this version run on my computer for a few days I released it to the public, starting the qmail beta test.

On 1 August 1996 I released qmail 0.90, 105044 words, ending the qmail beta test. On 20 February 1997 I released

*Date of this document: 2007.11.01. Permanent ID of this document: acd21e54d527dabf29afac0a2ae8ef41.

CSAW'07, November 2, 2007, Fairfax, Virginia, USA.

https://www.doc.ic.ac.uk/~livshits/classes/CO445H/reading/qmail.pdf

2) Defense in depth

Defense-In-Depth: Password Security Example

- Sys admins can require users to choose **strong passwords** to prevent guessing attacks
- To detect, can monitor server logs for large # of failed logins coming from an IP address and mark it as suspicious
- Contain by denying logins from suspicious IPs or require additional checks (e.g. cookies)
- To recover, monitor accounts that may have been hacked, deny suspicious transactions

3) Securing the Weakest Link

Securing the Weakest Link

- One-third of users choose a password that could be found in the dictionary
- Attacker can employ a dictionary attack and will eventually succeed in guessing someone’s password
- By using Least Privilege, can at least **mitigate** damage from compromised accounts

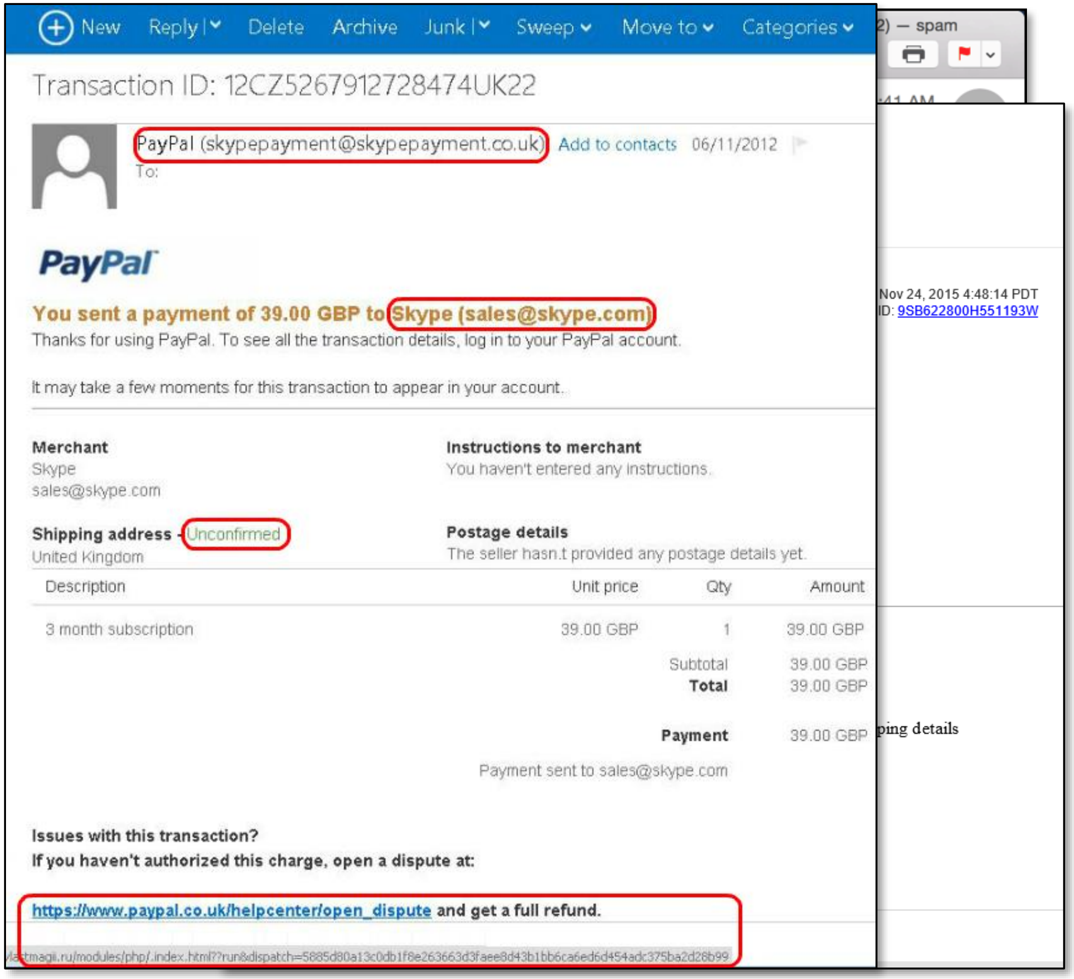
Social Engineering Attacks

Why employees are a businesses weakest link - and how to remedy that

employees are as liable as ever to fall for hacking scams and phishing schemes, mostly distributed via e-mail messages with rogue attachments or links

If passwords are so weak, why do so many companies rely on them? It is often a matter of "this is what we are used to", that prevents implementing a new system considered difficult and onerous

“People are our greatest asset,” proclaim companies all across the land – but that motto would perhaps be most appropriate to Hacker Incorporated, the loosely-affiliated organisation of cyber-baddies that has made a very successful business of invading computers and networks, for fun and great profit. According to Statista, there were over 1,000 data breaches which compromised 1.9 billion data records in 2016 – compared to just 784 in 2015.



New Reply | Delete Archive Junk | Sweep Move to Categories

Transaction ID: 12CZ5267912728474UK22

PayPal (skypepayment@skypepayment.co.uk) Add to contacts 06/11/2012

To:

PayPal

You sent a payment of 39.00 GBP to Skype (sales@skype.com)

Thanks for using PayPal. To see all the transaction details, log in to your PayPal account.

It may take a few moments for this transaction to appear in your account.

Merchant
Skype
sales@skype.com

Instructions to merchant
You haven't entered any instructions.

Shipping address - Unconfirmed
United Kingdom

Postage details
The seller hasn.t provided any postage details yet.

Description	Unit price	Qty	Amount
3 month subscription	39.00 GBP	1	39.00 GBP
		Subtotal	39.00 GBP
		Total	39.00 GBP
		Payment	39.00 GBP

Payment sent to sales@skype.com

Issues with this transaction?
If you haven't authorized this charge, open a dispute at:

https://www.paypal.co.uk/helpcenter/open_dispute **and get a full refund.**

/astmagii.ru/modules/php/.index.html??run&dispatch=5885d80a13c0db1f8e263663d3faee8d43b1bb6ca6ed6d454adc375ba2d28b99

2) – spam

Nov 24, 2015 4:48:14 PDT
ID: 9SB622800H551193W

ping details

Password Cracking Tool

Not all passwords can be recovered in a reasonable time using these approaches. If you have difficulties, use the guaranteed password reset function from commercial software.

Back-Doors

- Malicious developers (aka *insider threats*)
  - Can put back doors into their programs
  - Should employ code review
  - Or static analysis
- Untrustworthy libraries
- Is open source better here?

Wipeout: When Your Company Kills Your iPhone

by MARTIN KASTE

November 22, 2010 3:19 PM ET

Listen to the Story

All Things Considered

A personal iPhone can be set up to receive company e-mail via a Microsoft Exchange Server. But once it is set up, the phone can receive a variety of commands from the server including a remote wipe, which can destroy all the data and disable the phone.

A few weeks ago, Amanda Stantor

She had been talking on it and nav
Then, without any warning or error

Everything was gone — all her cor

It was only after she got home to S
killed by her employer, a publishin

Destruction Via E-Mail

Someone in the IT department hac
destruct command that's delivered
wouldn't have been surprised to se

But this iPhone was hers.

16

4) Fail-Safe

Fail-Safe Stance

- Expect & Plan for System Failure
- Common world example: lifts in buildings
  - Designed with expectation of power failure
  - In power outage, can grab onto cables or guide rails
- Ex: If firewall fails, let no traffic in
  - Deny access by default
  - Don’t accept all (including malicious), because that gives attacker additional incentive to cause failure

Fail Safely, Not Like This


```
isAdmin = true;
try {
  codeWhichMayFail();
  isAdmin = isUserInRole( “Administrator” );
  ...
} catch (Exception ex) {
  log.write(ex.toString());
}
```

5) Avoid Security through Obscurity

Security Through Obscurity

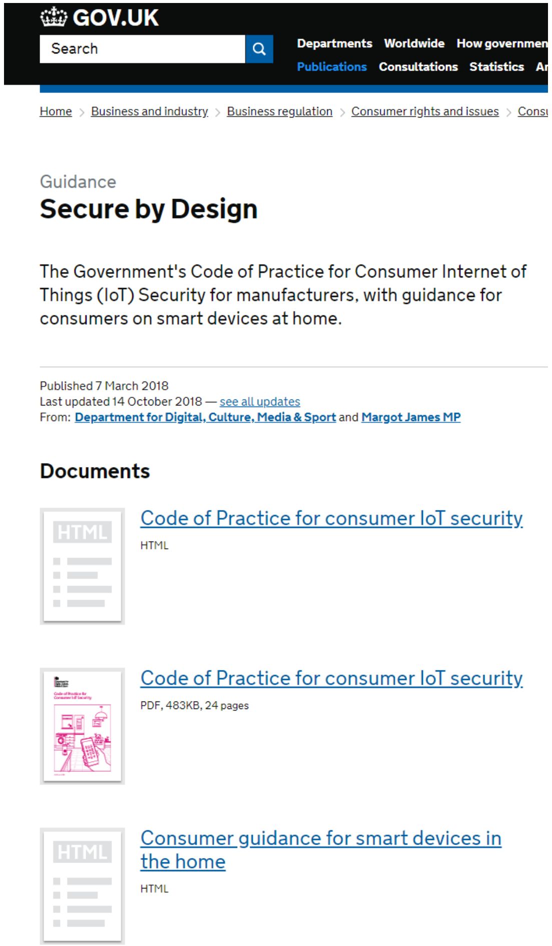
- Security Through Obscurity (STO) is the belief that a system of any sort can be secure so long as nobody outside of its implementation group is allowed to find out anything about its internal mechanisms.
- Hiding account passwords in binary files or scripts with the presumption that "nobody will ever find it" is a prime case of STO.
- Security through obscurity would be **burying your money** under a tree.
- The **only thing** that makes it safe is no one knows it's there.
- Real security is putting it behind a **lock** or **in a safe**.
- You can **put the safe on the street corner** because what makes it secure is that no one can get inside it but you.

21

6) Secure by default

Secure by Design

- The guidelines within the Code of Practice include:
  - Ensuring that IoT devices do not contain default passwords.
  - Defining and implementing vulnerability disclosure policy.
  - Ensuring software for devices is regularly updated.



National Cyber Security Centre

National Cyber Security Centre
a part of GCHQ

We are using cookies to give you the best experience on our site. By continuing to use our website without changing the settings, you are agreeing to our use of cookies.

Guidance | Threats | Incident Management | Marketplace | Education & Research | Insi

Blogs | Events | Case studies

Home > Insight > Blog

Blog post

Open sourcing MailCheck

Created: 21 Feb 2017
Updated: 21 Feb 2017
Author: Richard C
Part of: Secure by default, Digital services

Make things open, it makes them better

Picture credit: Paul Downey, Open Government License

NCSC's new London-based headquarters wasn't the only thing we opened up last week, we also took the step of open sourcing the code behind one of our Active Cyber Defence projects. MailCheck is the tool we're building to support our work on securing government email, and reducing phishing of public sector brands. It's the technology behind the work mentioned in our previous blog, Making email mean something again.

It's early days for MailCheck, but we're keen to be transparent about what we're doing. The work so far has focused on building a processing engine for aggregate DMARC reports to help us understand the extent of spoofing of public sector domains. Future releases will provide dashboards and tools to help public sector organisations understand the security configuration of their email services, as well as make sense of DMARC reports to take appropriate action. We'll open source these new features as we add them.

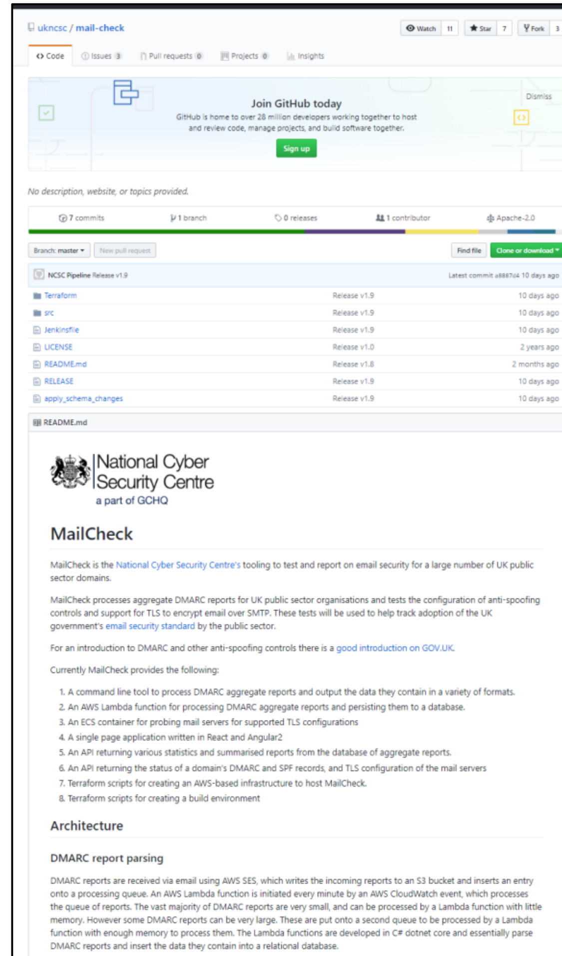
For the uninitiated, DMARC stands for Domain-based Messaging and Reporting Conformance. It's essentially a protocol you apply to your domains via DNS to help mail servers figure out whether email they receive claiming to be from one of your domains is legitimate or not. There is a great introduction to DMARC on the GOV.UK website, along with the email security standard we worked on with the Government Digital Service.

For the organisations looking to implement DMARC - and we hope this will be any of them with a brand that needs protecting - it's a simple matter of setting a few DNS records. The hard part is in making sense of the reports, and it's our goal to take the pain away for public sector organisations trying to do this.

Whilst it's early days for the project so far, the code we've produced has been built to scale. We're already processing DMARC reports for around 500 domains a week and believe the design we have will scale significantly. However if you're working with DMARC on smaller projects, then some of the tools contained within our code base may still be of use, particularly our basic command line tool to process a folder of DMARC reports into different formats.

We expect to open source a number of other projects in the coming months on our GitHub account - we'll let you know when we do.

You'll find the MailCheck source code at https://github.com/ukncsc/mail-check.



ukncsc / mail-check

Watch 11 | Star 7 | Fork 3

Code | Issues 3 | Pull requests 0 | Projects 0 | Insights

Join GitHub today

GitHub is home to over 28 million developers working together to host and review code, manage projects, and build software together.

Sign up

Dismiss

No description, website, or topics provided.

7 commits | 1 branch | 0 releases | 1 contributor | Apache-2.0

Branch: master | New pull request | Find file | Clone or download

NCSC Pipeline Release v1.9 — Latest commit a8887d4 10 days ago

Terraform	Release v1.9	10 days ago
src	Release v1.9	10 days ago
Jenkinsfile	Release v1.9	10 days ago
LICENSE	Release v1.0	2 years ago
README.md	Release v1.8	2 months ago
RELEASE	Release v1.9	10 days ago
apply_schema_changes	Release v1.9	10 days ago

README.md

National Cyber Security Centre
a part of GCHQ

MailCheck

MailCheck is the National Cyber Security Centre's tooling to test and report on email security for a large number of UK public sector domains.

MailCheck processes aggregate DMARC reports for UK public sector organisations and tests the configuration of anti-spoofing controls and support for TLS to encrypt email over SMTP. These tests will be used to help track adoption of the UK government's email security standard by the public sector.

For an introduction to DMARC and other anti-spoofing controls there is a good introduction on GOV.UK.

Currently MailCheck provides the following:

1. A command line tool to process DMARC aggregate reports and output the data they contain in a variety of formats.
2. An AWS Lambda function for processing DMARC aggregate reports and persisting them to a database.
3. An ECS container for probing mail servers for supported TLS configurations
4. A single page application written in React and Angular2
5. An API returning various statistics and summarised reports from the database of aggregate reports.
6. An API returning the status of a domain's DMARC and SPF records, and TLS configuration of the mail servers
7. Terraform scripts for creating an AWS-based infrastructure to host MailCheck.
8. Terraform scripts for creating a build environment

Architecture

DMARC report parsing

DMARC reports are received via email using AWS SES, which writes the incoming reports to an S3 bucket and inserts an entry onto a processing queue. An AWS Lambda function is initiated every minute by an AWS CloudWatch event, which processes the queue of reports. The vast majority of DMARC reports are very small, and can be processed by a Lambda function with little memory. However some DMARC reports can be very large. These are put onto a second queue to be processed by a Lambda function with enough memory to process them. The Lambda functions are developed in C# dotnet core and essentially parse DMARC reports and insert the data they contain into a relational database.

Key Design Principles and Patterns

- Avoid **elevated privileges**
- Use **layered defense** (prevention, detection, containment, and recovery)
- Secure the **weakest links**
- Have **fail-safes**, i.e. crash gracefully
- Don’t trust in **Security through Obscurity**
- Make design that is secure **by default**, out of the box, without the need to do anything

Languages and Vulnerabilities

- Most vulnerabilities are the result of unsafe programming and unsafe programming practices and patterns
- Languages can do a lot to improve things
- We will look at Java language design
- We will look at advanced platform protections and their failings

Language design: C++ vs. Java

- Manual memory allocation and deallocation
- Pointer arithmetic and casts
- Focus on speed and hardware control

- Memory allocation is largely automatic
- Type safety
- Checking array bounds
- Performance suffers somewhat

Java Security Basics

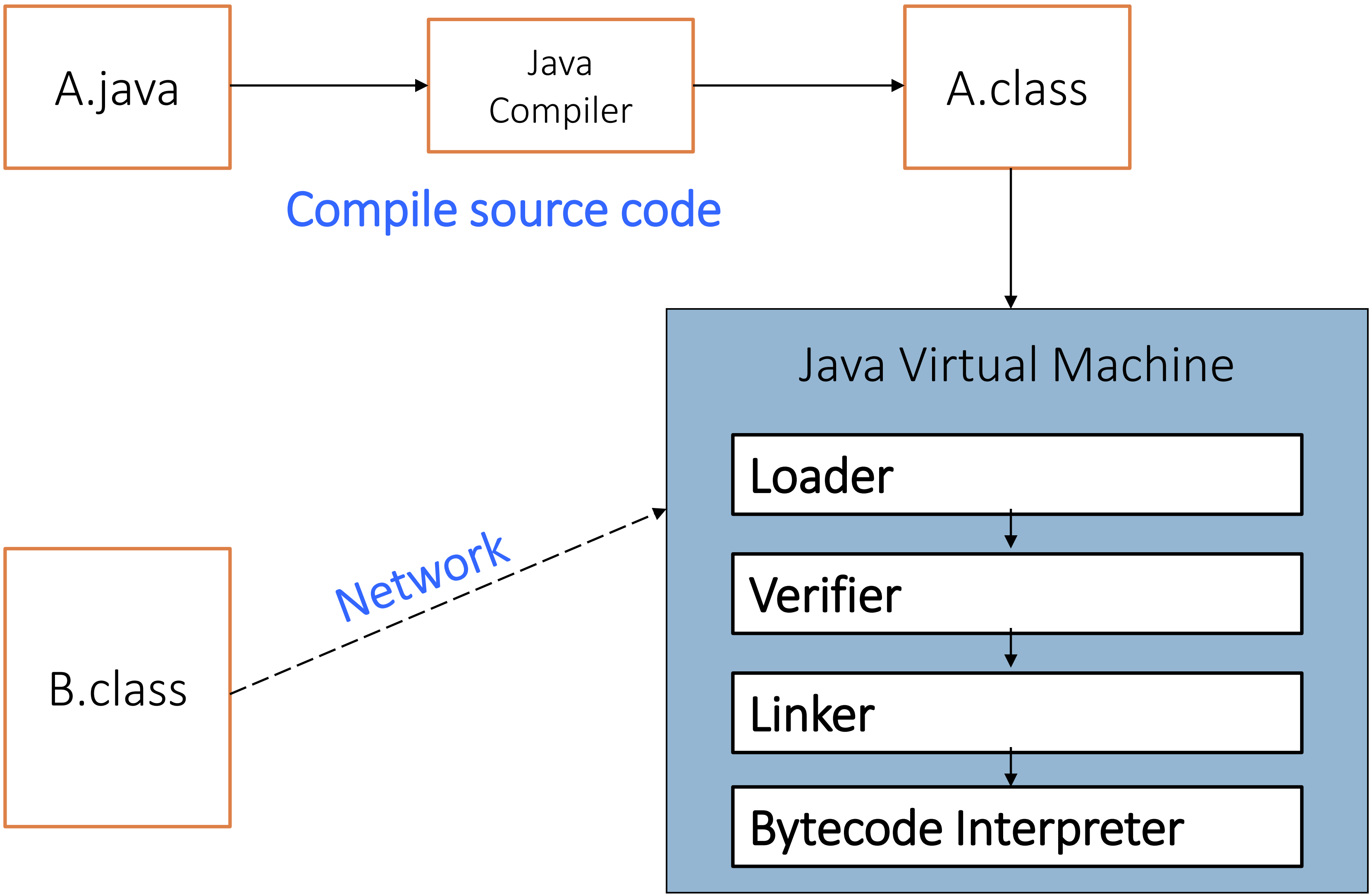
(based on slides from John Mitchell)

Java Implementation Principles

- Compiler and Virtual Machine
  - Compiler produces bytecode
  - Virtual machine loads classes on demand, verifies bytecode properties, interprets bytecode

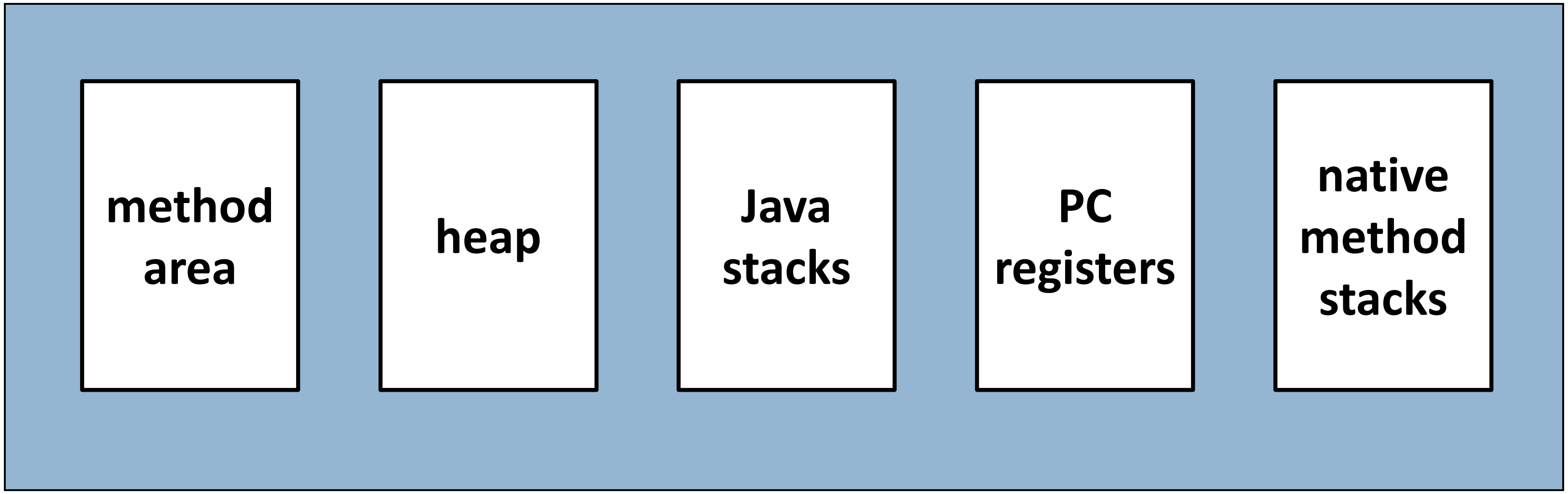
- Why this design?
  - Bytecode interpreter/compilers used before
    - Pascal “pcode”
    - Smalltalk compilers use bytecode
  - Minimize machine-dependent part of implementation
    - Do optimization on bytecode when possible
    - Keep bytecode interpreter simple
  - For Java, this gives portability
    - Transmit bytecode across network

Java Virtual Machine Architecture



JVM Memory Areas

- Java program has one or more threads
- Each thread has its own stack
- All threads share same heap



Class Loaders

- Runtime system loads classes as needed
- When class is referenced, loader searches for file of compiled bytecode instructions
- Default loading mechanism can be replaced
- Define **alternate** ClassLoader object
  - Extend the abstract ClassLoader class and implementation
  - ClassLoader does not implement abstract method loadClass, but has methods that can be used to implement **loadClass**

JVM Linker and Verifier

- **Linker**
  - Adds compiled class or interface to runtime system
  - Creates static fields and initializes them
  - Resolves names
  - Checks symbolic names and replaces with direct references
- **Verifier**
  - Check bytecode of a class or interface before loaded
  - Throw VerifyError exception if error occurs

Verifier Design

- Bytecode **may not** come from standard compiler
  - Evil hacker may write dangerous bytecode

- Verifier checks **correctness** of bytecode
  - Every instruction must have a valid operation code
  - Every branch instruction must branch to the start of some other instruction, not middle of instruction
  - Every method must have a structurally correct signature
  - Every instruction obeys the Java type discipline
  - This is fairly complicated and tricky

Verifier Issues: CVE-2012-1723

- **CVE-2012-1723:** This is a vulnerability in the HotSpot bytecode verifier that has been present since at least Java 1.4.
- Vulnerable version of the HotSpot compiler will perform an **invalid optimization** when verifying deferred GETSTATIC/PUTSTATIC/GETFIELD/PUTFIELD instructions (hereafter referred to as "field access instructions") in preparation of JIT-compiling a method
- To exploit this vulnerability, you need to craft a method with **at least two different field access instructions referring to the same field**, and have to force the method to be JITed while their verification is still deferred (i. e. you have to call the method a lot of times but make sure none of these executions touch those instructions, for example by passing a parameter that makes sure the method will end early in those executions). Then call it again for the effect.
- http://schierlm.users.sourceforge.net/CVE-2012-1723.html for more details

Towards Memory Safety

- Perform run-time checks such as **array bounds**
- All **casts** are checked to make sure type safe
- All **array references** are checked to make sure the array index is within the array bounds
- References are tested to make sure they are not **null** before they are dereferenced.
- No pointer arithmetic
- Automatic garbage collection

If program accesses memory, that memory is allocated to the program and declared with correct type

Java and Native Interactions

- Possible to compile bytecode class file to native code
- JITs are used for performance
- Java programs can call native methods, typically functions written in C
- C# and .NET take C/C++ interop very seriously

```
class PlatformInvokeTest {
    [DllImport("msvcrt.dll")]
    public static extern int puts(string c);
    [DllImport("msvcrt.dll")]
    internal static extern int _flushall();

    public static void Main()
    {
        puts("Test");
        _flushall();
    }
}
```

Java Security Mechanisms

- Sandboxing
  - Run program in restricted environment
  - Analogy: child's sandbox with only safe toys
  - This term refers to features of loader, verifier, interpreter that restrict program
- Code signing
  - Use cryptography to establish origin of class file
  - This info can be used by security manager
- Class loader
  - Separate namespaces for separate class loaders
  - Associates *protection domain* with each class
- Verifier and JVM run-time tests
  - NO unchecked casts or other type errors
  - NO buffer/array overflows
  - Preserves private, protected visibility levels
- Security Manager
  - Called by library functions to decide if request is allowed
  - Uses protection domain associated with code, user policy
  - Coming up in a few slides: stack inspection

Security Manager

- Java library functions call Security Manager

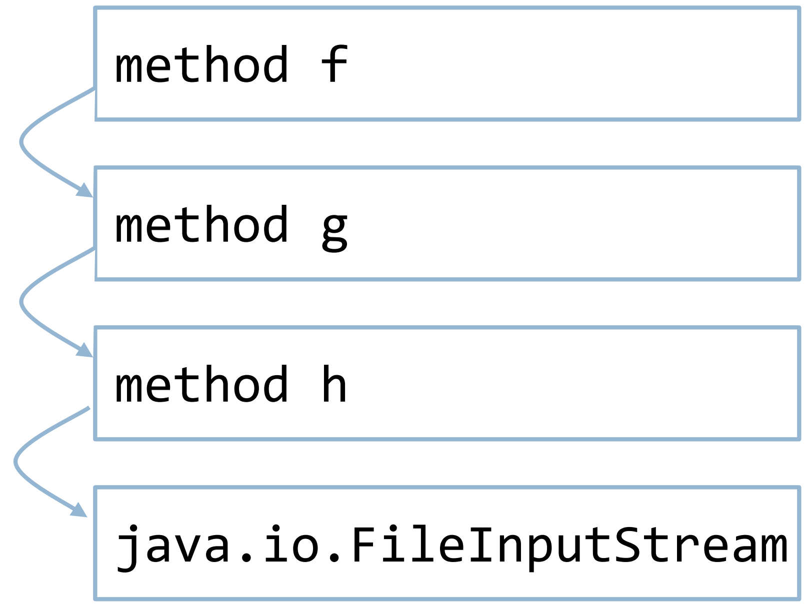
- Security manager object answers at run time
  - Decide if calling code is allowed to do operation
  - Examine protection domain of calling class
    - Signer: organization that signed code before loading
    - Location: URL where the Java classes came from
  - Uses the system policy to decide access permission

Sample Security Manager Methods

checkExec	Checks if the system commands can be executed.
checkRead	Checks if a file can be read from.
checkWrite	Checks if a file can be written to.
checkListen	Checks if a certain network port can be listened to for connections.
checkConnect	Checks if a network connection can be created.
checkCreate ClassLoader	Check to prevent the installation of additional ClassLoaders.

Stack Inspection

- Permission depends on
  - Permission of calling method
  - Permission of all methods above it on stack
  - Up to method that is trusted and asserts this trust



Java: Things Didn't Quite Go According to Plan

Java's popularity among developers and widespread usage in Web browsers all but guarantees continuing interest from threat actors seeking new lines of attack. Malware authors have advanced quickly—not just finding new vulnerabilities, but developing clever ways to exploit them.

Multiple payload downloads in a single attack session have grown common, maximizing the profit potential from crimeware. Using .jar files themselves to carry malware payloads (as seen in the Cool exploit kit example) allows attackers to bundle multiple payloads with one attack and bypass detection.

Motivated by profits, cyber attackers are bound to adopt more intelligent exploit kits that "know their victim." That was the case in several recent attacks. These attacks used plugin-detection scripts and advanced exploit chains to evade discovery and compromise websites for drive-by malware downloads. Post-exploit, multi-stage malware downloads will continue to mushroom as more threat actors scramble for a piece of the crimeware pie.

1	CVE-2013-5812	2013-10-16	2014-02-06	6.4
	nspecified vulnerability in Oracle Java SE 7u40 and earlier, Java SE 6u60 and earlier, a onfidentiality and availability via unknown vectors related to Deployment.			
2	CVE-2013-5804	2013-10-16	2014-10-04	6.4
	nspecified vulnerability in Oracle Java SE 7u40 and earlier, Java SE 6u60 and earlier, J arlier allows remote attackers to affect confidentiality and integrity via unknown vectors			
3	CVE-2013-5783	2013-10-16	2014-10-04	6.4
	nspecified vulnerability in Oracle Java SE 7u40 and earlier, Java SE 6u60 and earlier, J ttackers to affect confidentiality and integrity via unknown vectors related to Swing.			
4	CVE-2013-3829	2013-10-16	2014-10-04	6.4
	nspecified vulnerability in the Java SE, Java SE Embedded component in Oracle Java SE nd Java SE Embedded 7u40 and earlier allows remote attackers to affect confidentiality			
5	CVE-2013-2467	2013-06-18	2014-01-07	6.9
	nspecified vulnerability in the Java Runtime Environment (JRE) component in Oracle Ja nd availability via unknown vectors related to the Java installer.			
6	CVE-2013-2439	2013-04-17	2013-12-05	6.9
	nspecified vulnerability in the Java Runtime Environment (JRE) component in Oracle Ja nd JavaFX 2.2.7 and earlier allows local users to affect confidentiality, integrity, and ava			
7	CVE-2013-2407	2013-06-18	2014-10-04	6.4
	nspecified vulnerability in the Java Runtime Environment (JRE) component in Oracle Ja emote attackers to affect confidentiality and availability via unknown vectors related to ommented on claims from another vendor that this issue is related to "XML security and			
8	CVE-2013-0432	2013-02-01	2014-10-04	6.4
	nspecified vulnerability in the Java Runtime Environment (JRE) component in Oracle Ja .4.2_40 and earlier, and OpenJDK 6 and 7, allows remote attackers to affect confidenti e February 2013 CPU. Oracle has not commented on claims from another vendor that			
9	CVE-2013-0430	2013-02-01	2013-11-02	6.9
	nspecified vulnerability in the Java Runtime Environment (JRE) component in Oracle Ja onfidentiality, integrity, and availability via unknown vectors related to the installation p			
10	CVE-2012-5071	2012-10-16	2014-10-04	6.4
	nspecified vulnerability in the Java Runtime Environment (JRE) component in Oracle Ja lows remote attackers to affect confidentiality and integrity, related to JMX.			
11	CVE-2012-4416	2012-10-16	2014-10-04	6.4
	nspecified vulnerability in the Java Runtime Environment (JRE) component in Oracle Ja affect confidentiality and integrity via unknown vectors related to Hotspot.			
12	CVE-2012-0502	2012-02-15	2014-10-04	6.4
	nspecified vulnerability in the Java Runtime Environment (JRE) component in Oracle Ja nd 1.4.2_35 and earlier allows remote untrusted Java Web Start applications and untru			
13	CVE-2011-3563	2012-02-	2014-10-	6.4

Analyzing Java Exploits

An In-Depth Study of More Than Ten Years of Java Exploitation

Philipp Holzinger[1], Stefan Triller[1], Alexandre Bartel[2], and Eric Bodden[3,4]
[1]Fraunhofer SIT, [2]Technische Universität Darmstadt, [3]Paderborn University, [4]Fraunhofer IEM
[1]{firstname.lastname}@sit.fraunhofer.de, [2]alexandre.bartel@cased.de
[3]eric.bodden@uni-paderborn.de

ABSTRACT

When created, the Java platform was among the first runtimes designed with security in mind. Yet, numerous Java versions were shown to contain far-reaching vulnerabilities, permitting denial-of-service attacks or even worse allowing intruders to bypass the runtime's sandbox mechanisms, opening the host system up to many kinds of further attacks.

This paper presents a systematic in-depth study of 87 publicly available Java exploits found in the wild. By collecting, minimizing and categorizing those exploits, we identify their commonalities and root causes, with the goal of determining the weak spots in the Java security architecture and possible countermeasures.

Our findings reveal that the exploits heavily rely on a set of nine weaknesses, including unauthorized use of restricted classes and confused deputies in combination with caller-sensitive methods. We further show that all attack vectors implemented by the exploits belong to one of three categories: single-step attacks, restricted-class attacks, and information hiding attacks.

The analysis allows us to propose ideas for improving the security architecture to spawn further research in this area.

1. INTRODUCTION

From a security point of view, a virtual machine's goal is to contain the execution of code originating from untrusted sources in such a way that it cannot impede the security goals of the host machine. For instance, the code should not be able to access sensitive information to which access has not been granted, nor should it be able to launch a denial-of-service attack. Many virtual machines try to contain untrusted code through a so-called sandbox model. Conceptually, a sandbox runs the untrusted code in a controlled environment, by separating its execution and its data from that of trusted code, and by allowing it only to have access to a limited and well-defined set of system resources.

This paper investigates more than ten years of insecurities and exploitation of the Java platform, whose security concepts rely heavily on such sandbox model. Conceptually, the Java Runtime Environment (JRE) uses a sandbox to contain code whose origin is untrusted in a restricted environment. When executing a Java applet from an untrusted site within a browser, its access is controlled. Sandboxed applets are only allowed to perform a very limited set of tasks such as making network connections to the host they were loaded from, or display HTML documents.[1] A second use case of sandboxing in Java is on the server side: application servers use the sandbox mechanisms to isolate from one another and from the host systems the applications they serve.

While conceptually easy to grasp, the Java sandbox is actually anything but a simple box. Instead it comprises one of the world's most complex security models in which more than a dozen dedicated security mechanisms come together to—hopefully—achieve the desired isolation. To just give some examples: bytecode verification must prevent invalid code from coming to execution, access control must correctly tell apart trusted from untrusted code, and to prevent the forging of pointers type checking must in all cases properly distinguish pointer references from numeric values. As a consequence, the "sandbox" is only as good as the joint security architecture and implementation of all those different mechanisms that comprise the sandbox. Adding to that, the code implementing the sandbox has evolved over far more than a decade, involving dozens of developers, with virtually none of the original creators remaining in charge today, and with the lead maintenance of Java moving from Sun Microsystems to Oracle Inc. When considering all this, it may come as less of a surprise that over the years the Java runtime has seen a large number of devastating vulnerabilities and attacks, most of which lead to a full system compromise: an attacker would be able to inject and execute any code she desires, at the very least with the operating-system privileges assigned to the user running the Java virtual machine [7, 8]. Security vulnerabilities are present in different parts of the complex sandbox mechanism, involving issues such as type confusion, deserialization issues, trusted method chaining or confused deputies.

With Java being a runtime deployed on literally billions of devices, it is one of the most prevalent software systems in use today. Hence naturally, Java vendors such as Oracle and IBM are eager to fix vulnerabilities once they become known. But over the past years, the crafting of exploits by attackers and the crafting of patches by vendors has become a continuing arms race. Oftentimes, security patches lit-

[1]https://docs.oracle.com/javase/tutorial/deployment/applet/security.html

To summarize, this paper makes the following contributions:

- a collection of 61 working Java exploits, based on a set of 87 original exploits,
- an analysis and categorization of the Java exploits in terms of intended behavior and primitives,
- an analysis of Java in terms of its weak spots with respect to security, and
- potential security fixes for those weak spots.

Weakness	# exploits
Unauthorized use of restricted classes (W5)	32 (52%)
Loading of arbitrary classes (W4)	31 (51%)
Unauth. definition of privil. classes (W6)	31 (51%)
Reflective access to methods and fields (W8)	28 (45%)
Confused deputies (W2)	22 (36%)
Caller sensitivity (W1)	22 (36%)
MethodHandles (W9)	21 (34%)
Serialization and type confusion (W7)	9 (15%)
Privileged code execution (W3)	7 (11%)

Table 2: Overview of the weaknesses we identified and the number of minimal exploits that use them. One exploit can use more than one weakness.

https://www.abartel.net/static/p/ccs2016-10yearsJavaExploits.pdf

Buffer Overflows: Attacks and Defenses for the Vulnerability of the Decade

Buffer Overflows: Attacks and Defenses for the Vulnerabilty of the Decade – Cowan et al. 2000

Some of you may recall reading "**Smashing the Stack for Fun and Profit**" (hard to believe that was published in 1996!), which helped to raise consciousness of buffer overflow attacks. In this paper from 2000 Cowan et al. provide a very readable breakdown of how they work and the potential defenses against them.

> *Buffer overflows have been the most common form of security vulnerability in the last ten years (1990-2000). More over, buffer overflow vulnerabilities dominate in the area of remote network penetration vulnerabilities, where an anonymous Internet user seeks to gain partial or total control of a host. Because these kinds of attacks enable anyone to take total control of a host, they represent one of the most serious classes security threats.*

Defending Against Buffer Overflows

There are four basic mechanisms of defense against buffer overflow attacks: writing correct programs; enlisting the help of the operating system to make storage areas for buffers non-executable; enhanced compilers that perform bounds checking; and performing integrity checks on code pointers before dereferencing them.

"Writing correct code is a laudable but highly expensive proposition:"

> *Despite a long history of understanding of how to write secure programs vulnerable programs continue to emerge on a regular basis... Even defensive code that uses safer alternatives such as strncpy and snprintf can contain buffer overflow vulnerabilities if the code contains an elementary off-by-one error. For instance, the lprm program was found to have a buffer overflow vulnerability, despite having been audited for security problems such as buffer overflow vulnerabilities.*

https://blog.acolyer.org/2015/09/18/buffer-overflows-attacks-and-defenses-for-the-vulnerability-of-the-decade/

Back to Native Code…

- Buffer overruns: Stack, Heap

hbo.c:

```
#include <stdio.h>
#include <string.h>
#define BUFSIZE 8
int main()
{
    unsigned long diff;
    char *buf1, *buf2;

    buf1 = (char *) malloc(BUFSIZE);
    buf2 = (char *) malloc(BUFSIZE);
    diff = (unsigned long) buf2 - (unsigned long) buf1;
    printf("buf1 = %p, buf2 = %p, diff = 0x%x bytes¥n", buf1, buf2, diff);
    memset(buf2, 'A', BUFSIZE - 1);
    buf2[BUFSIZE - 1] = '¥0';
    printf("before overflow: buf2 = %s¥n", buf2);
    memset(buf1, 'B', (unsigned int) (diff + 3));
    printf("after overflow: buf2 = %s¥n", buf2);
    printf("after overflow: buf1 = %s¥n", buf1);
    return 0;
}
```

実行結果 (FreeBSD-4.10)

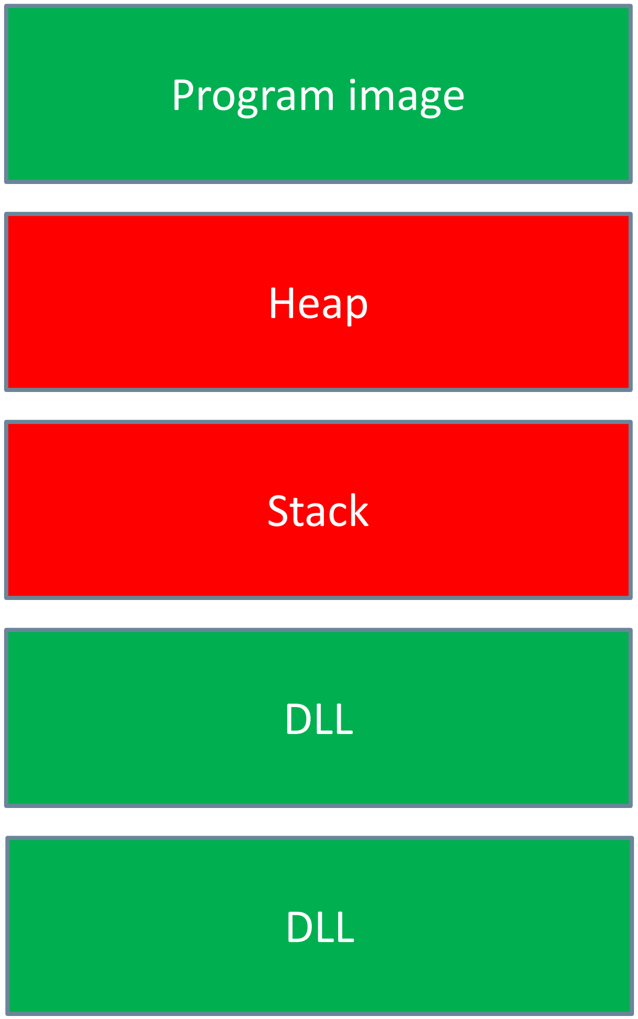
```
% cc -o hbo hbo.c
% ./hbo
buf1 = 0x804b030, buf2 = 0x804b040, diff = 0x10 bytes
before overflow: buf2 = AAAAAAA
after overflow: buf2 = BBBAAAA
after overflow: buf1 = BBBBBBBBBBBBBBBBBBBAAAA
```

DEP

- Hardware-enforced execution prevention technique
- Breaks the basics of memory exploitation
- Specifically, stacks and heaps become non-executable or NX
- So, can't **load** your shellcode there
- But... can jump to **existing** (shell-) code

Figure 7.14 Windows XP Service Pack 2 and Windows 2003

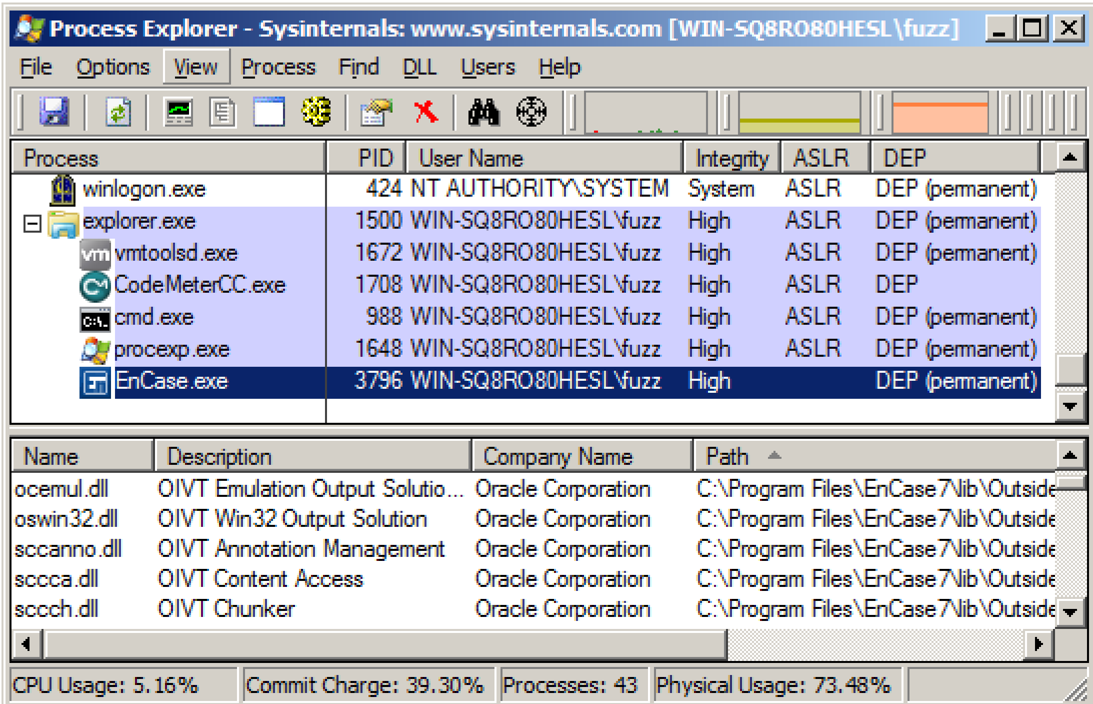
EIP Limitations



- Return-to-libc
- Pioneered in 1997
- EIP returns to an existing function
- Need control of the stack to place parameters there
- Typically, the stack is writeable

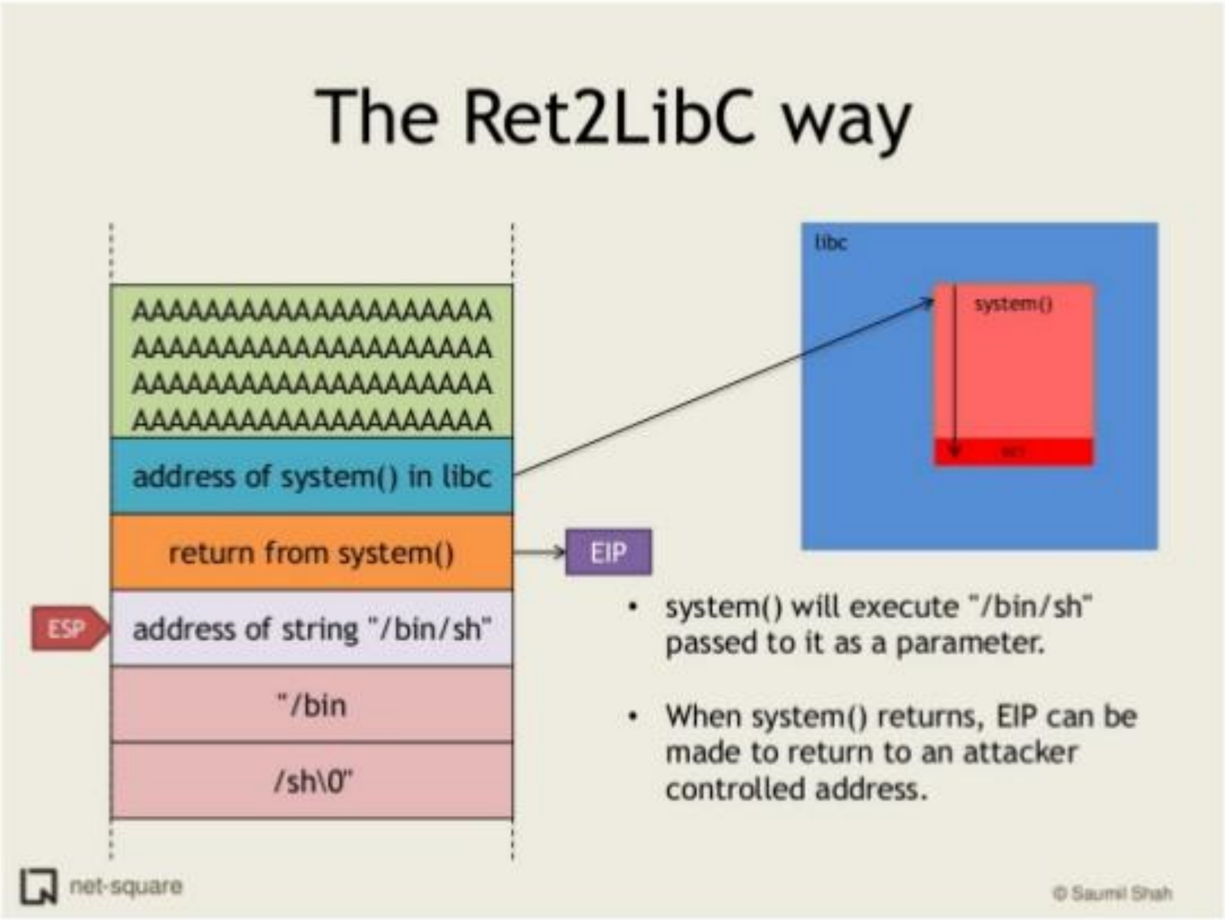
DEP and ASLR

Address space layout randomization (**ASLR**) is a memory-protection process for operating systems (OSes) that guards against buffer-overflow attacks by randomizing the location where system executables are loaded into memory.



https://www.zdnet.com/article/microsoft-says-aslr-behavior-in-windows-10-is-a-feature-not-a-bug/

Return-to-libc for system

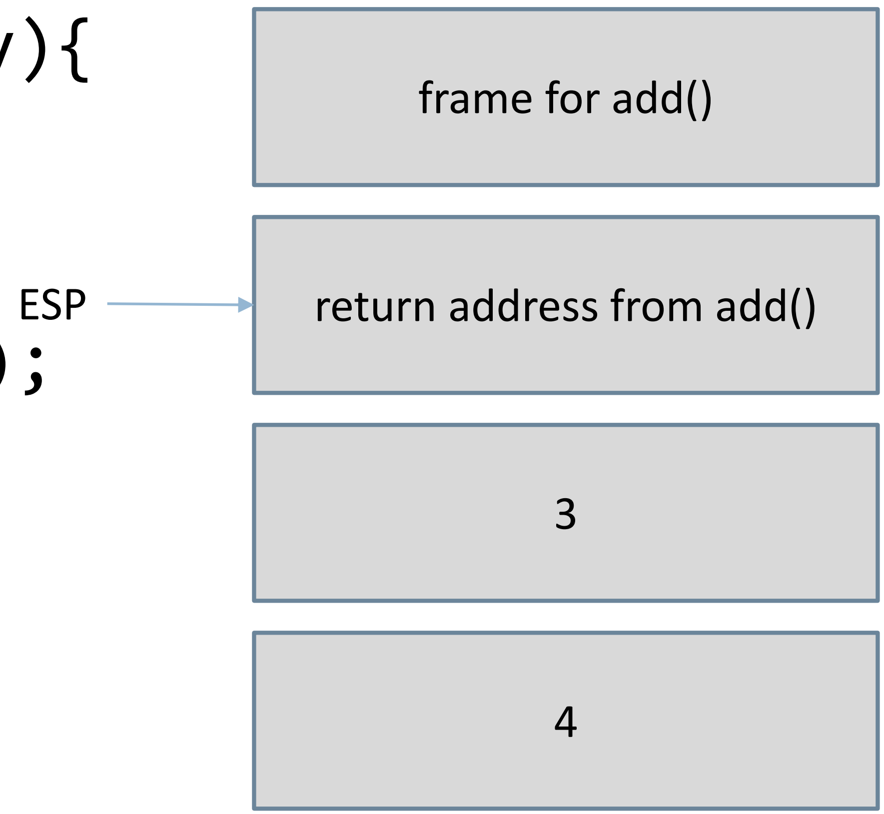


- It's possible to invoke an arbitrary function simply by placing a fake frame in stack memory
- It's possible to retain EIP control after the function return
- Ret2LibC forms the basis of return-oriented-programming

https://www.slideshare.net/saumilshah/dive-into-rop

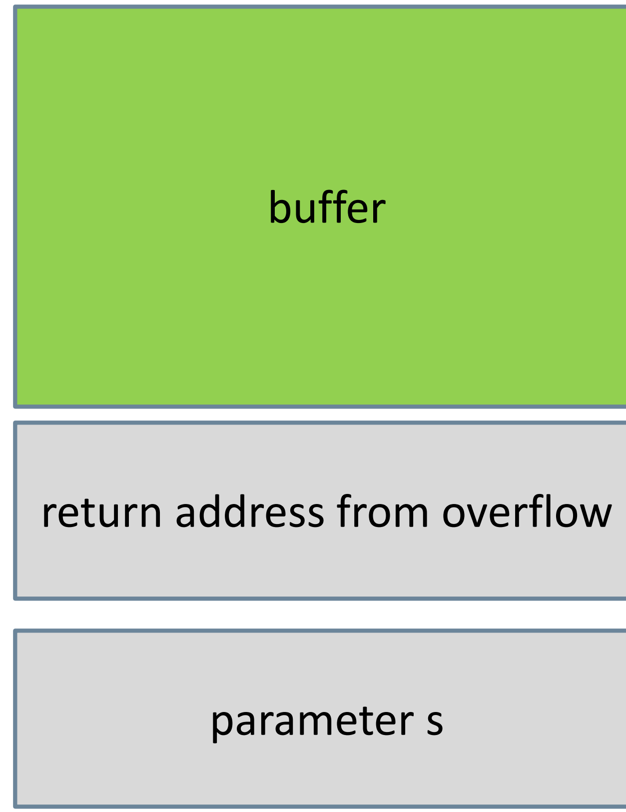
Function Calls


```
void add(int x, int y){
   int sum;
   sum = x+y;
   printf(“%d\n”, sum);
}

int main(){
   add(3,4);
}
```



Overflow the Buffer and Call add()


```
void overflow(char* s){
  char buffer[128];
  strcpy(buffer, s);
}

int main(){
  overflow(argv[1]);
}
```



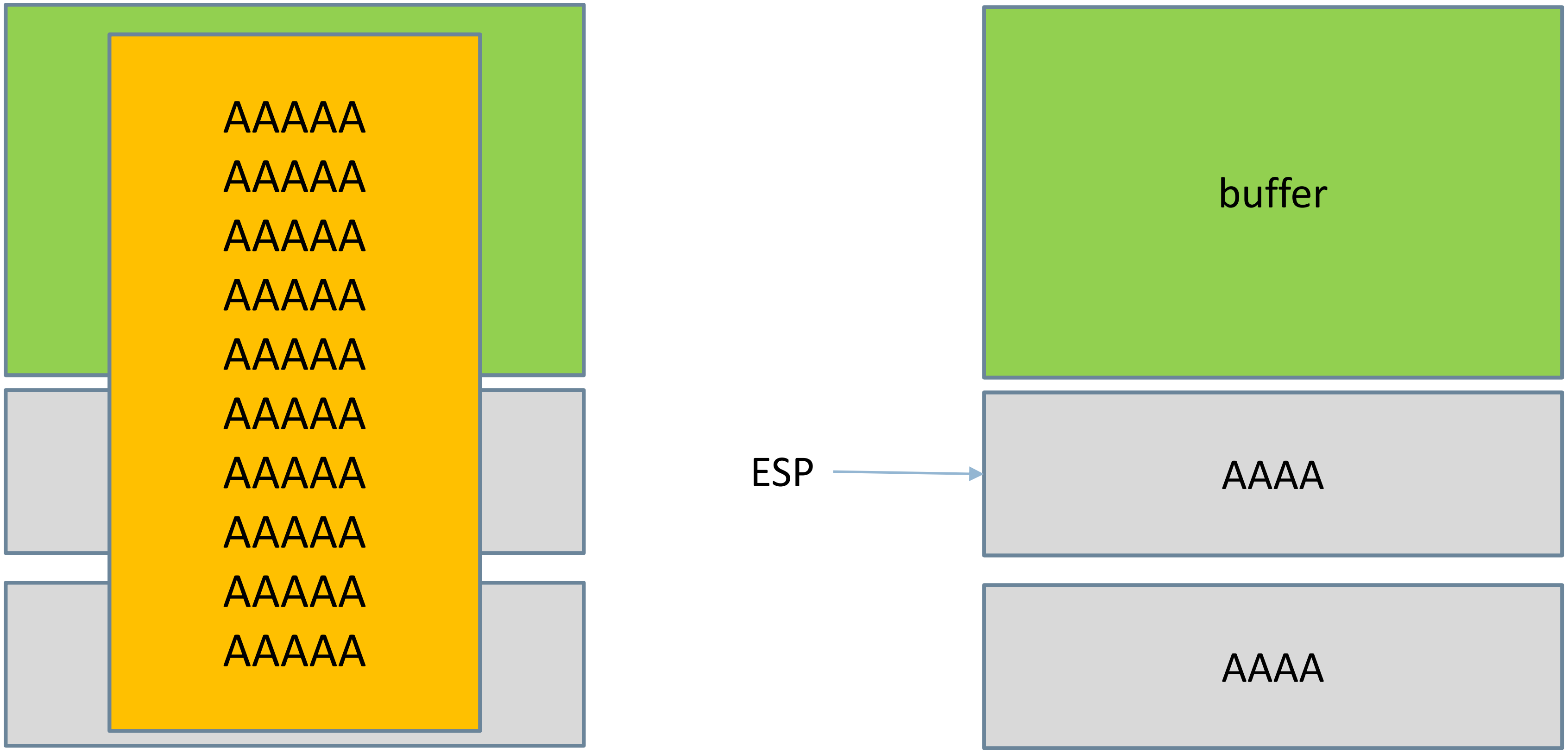
Calls and Returns

Call

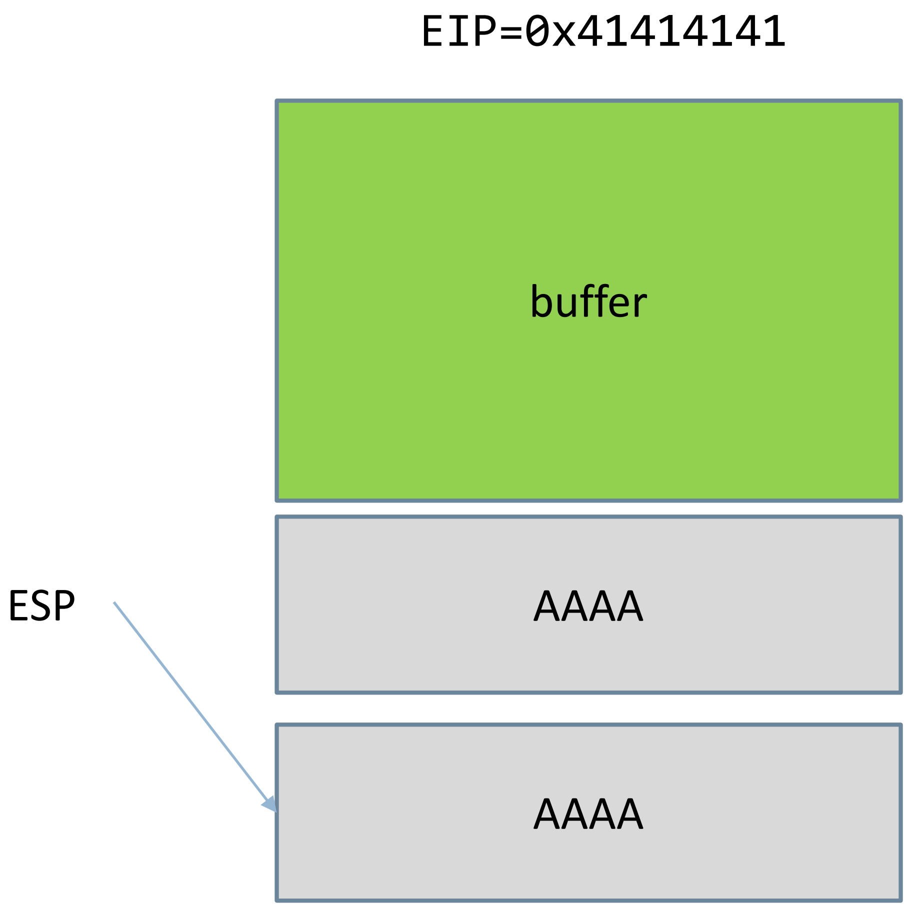
- push return address on the stack
- set up the stack
  - move ESP ahead
  - push EBP
  - mov ESP to EBP

- Function return
  - Leave
    - Restore EBP=POP EBP
    - MOV EBP to ESP
  - ret – return control back to the calling function
    - Return address stored earlier on the stack
    - POP EIP

Before the RET Instruction

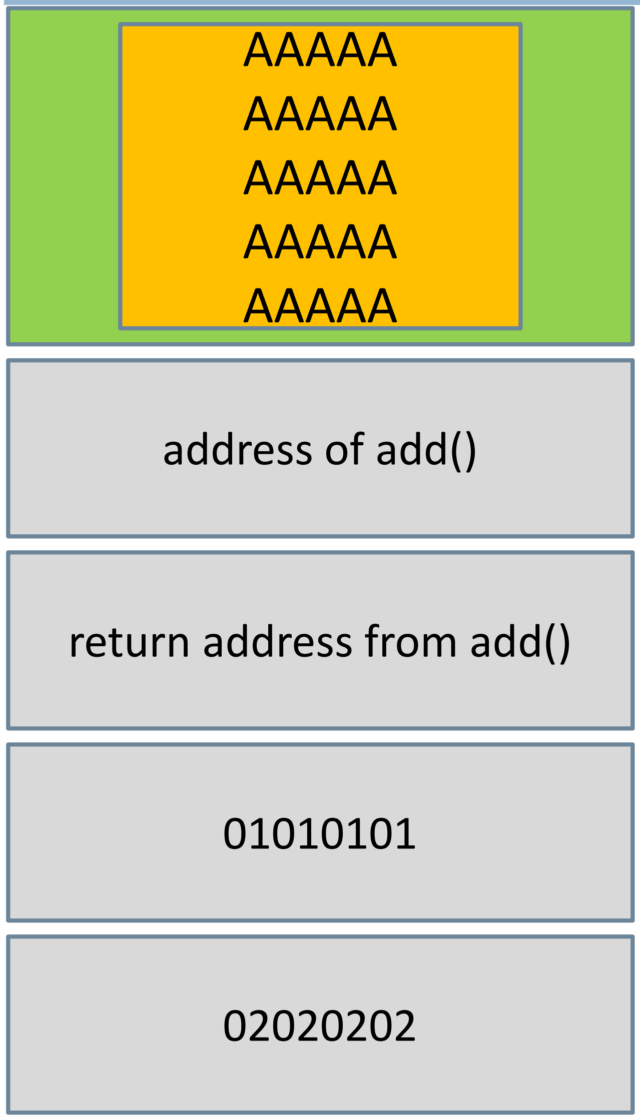


After the RET Instruction



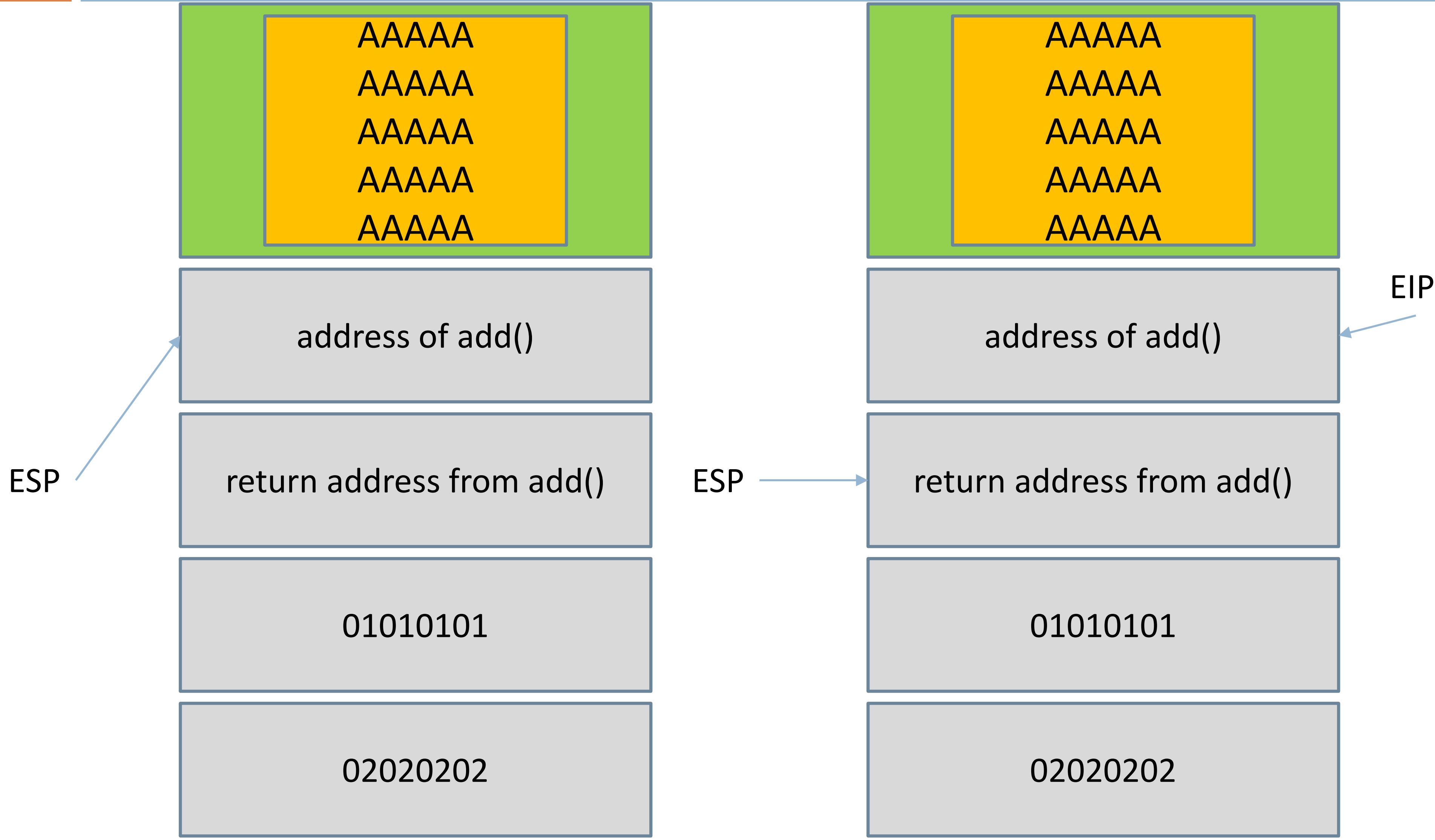
- To return to add()
  - Insert a fake frame in the buffer
  - Make overflow() return to add(01010101, 02020202)
  - What is the stack layout?

Calling add() Through overflow()

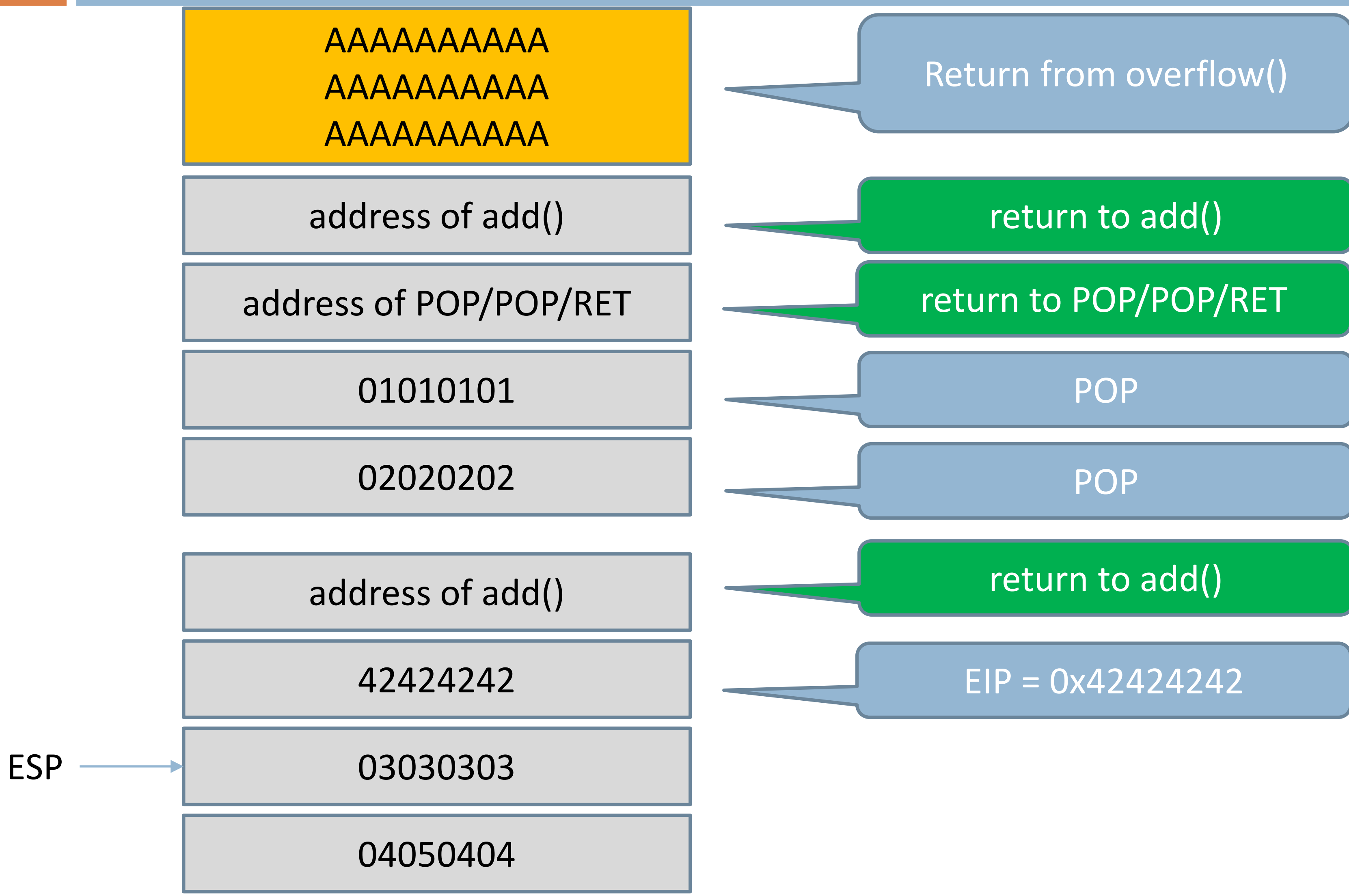


- By carefully crafting a frame
- We can have a program return to our **function of choice**
- We control the **parameters**
- We also control where to jump **after** the return

Before/after RET in overflow() Called



Chaining Multiple Function Calls



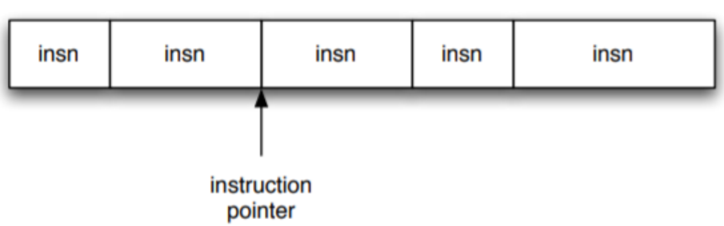
ROP Design Principles

- Piece together pieces of code
- Gadgets – primitive operations
- These are found in existing binaries to dodge DEP
- Can be the primary binary or the associated shared libraries
- Every gadget must end with RET (takes us to the next chained gadget)
- We find gadgets in function epilogues

EIP vs. ESP in ROP

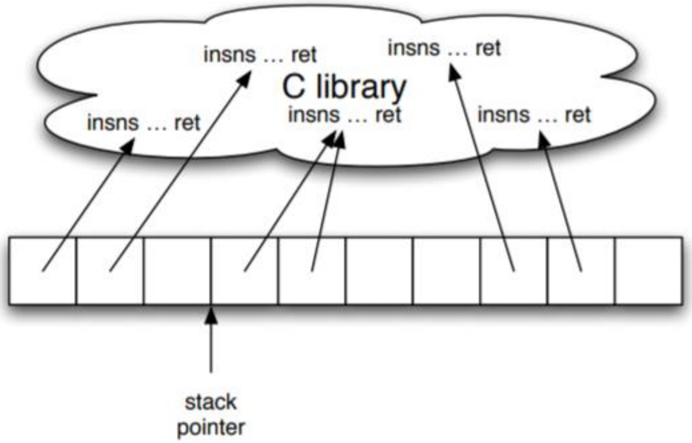
Classic EIP code

- N ops=N instructions
- EIP increments
- ESP fluctuates
- The CPU increments EIP automatically

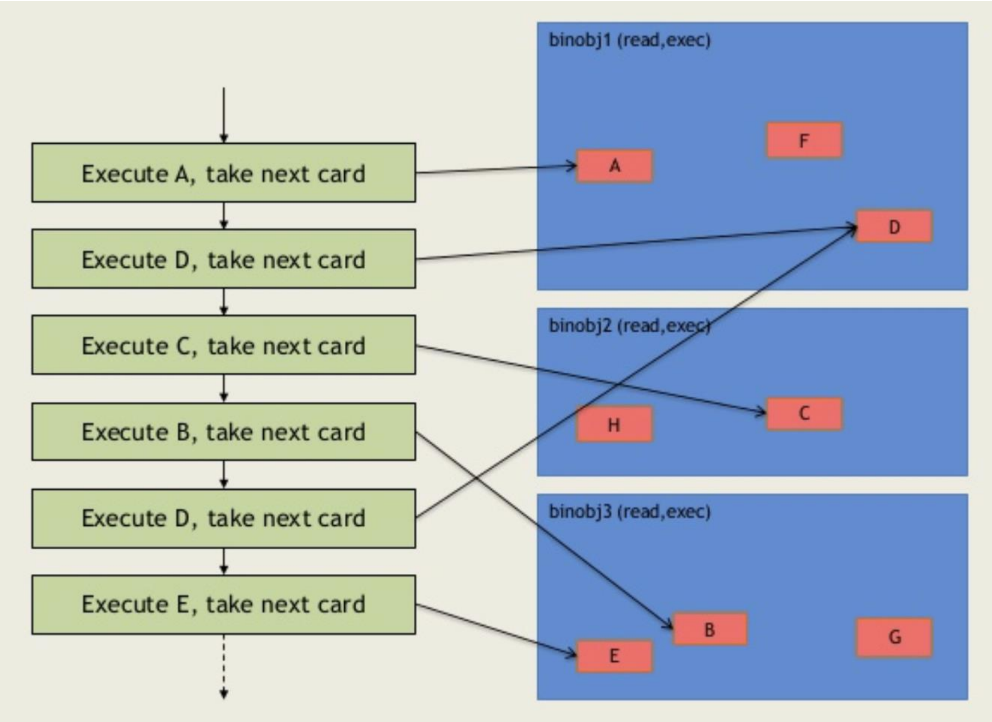


ROP code

- N ops=N frames
- ESP increments
- EIP fluctuates
- We control ESP via ret instructions

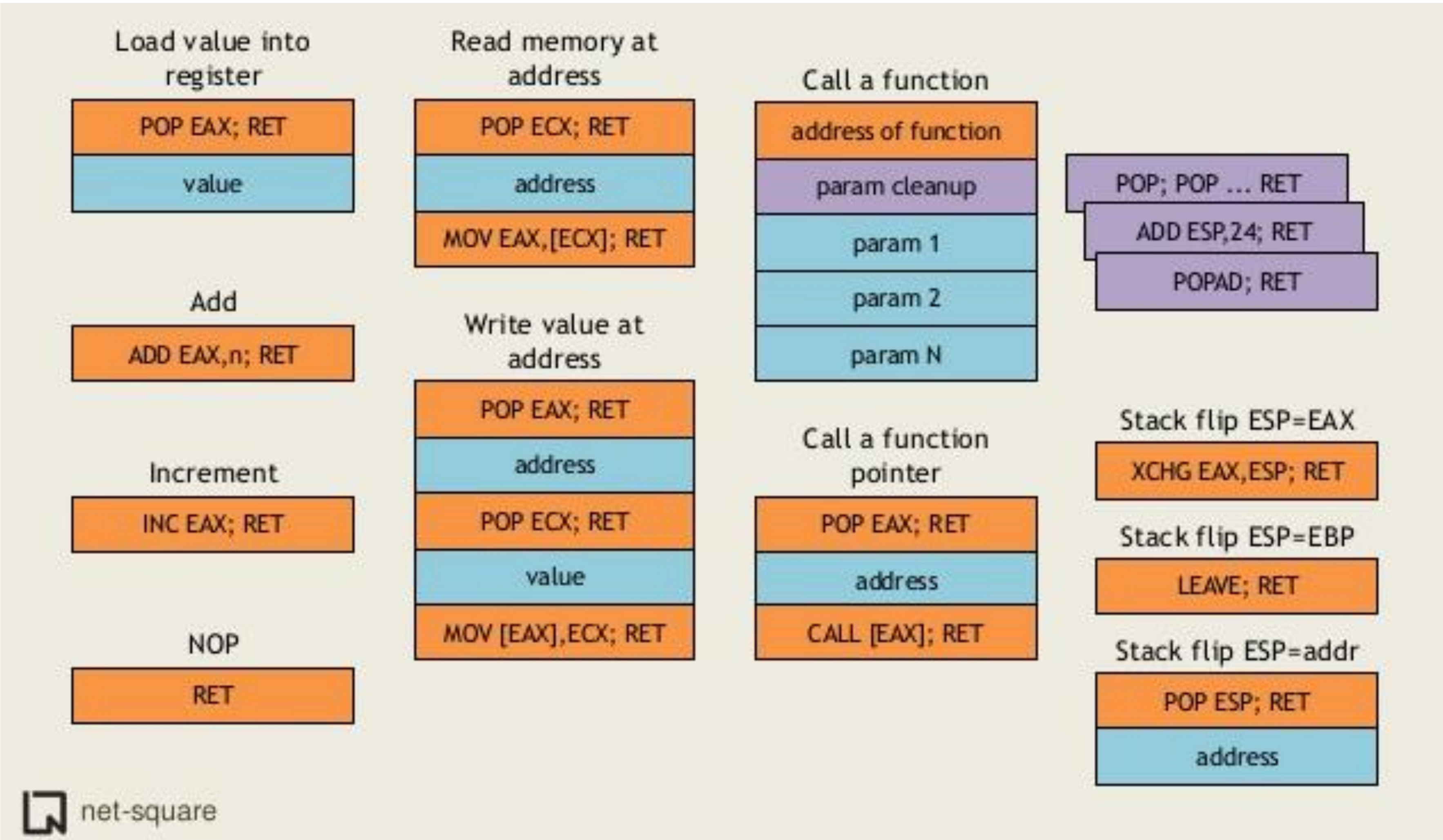


Gadgets Glued Together



https://www.slideshare.net/saumilshah/dive-into-rop

Gadget Dictionary



How to Find Gadgets?

- Disassemble code (binary + DLLs)
- Identify useful code sequences ending in ret as potential gadgets
- Assemble gadgets into desired shellcode
- *Return-Oriented Programming: Systems, Languages, and Applications* by Ryan Roemer, Erik Buchanan, Hovav Shacham and Stefan Savage
- Shacham et al. manually identified which sequences ending in ret in libc were useful gadgets
- Common shellcode was created with these gadgets.
- Everyone used libc, so gadgets and shellcode universal

Putting This All Together

- Several gadget compilers exist
- one example is ROPgadget on GitHub

```
0x0000000000440608 : mov dword ptr [rdx], ecx ; ret
0x00000000004598b7 : mov eax, dword ptr [rax + 0xc] ; ret
0x0000000000431544 : mov eax, dword ptr [rax + 4] ; ret
0x000000000045a295 : mov eax, dword ptr [rax + 8] ; ret
0x00000000004a3788 : mov eax, dword ptr [rax + rdi*8] ; ret
0x0000000000493dec : mov eax, dword ptr [rdx + 8] ; ret
0x00000000004a36f7 : mov eax, dword ptr [rdx + rax*8] ; ret
0x0000000000493dc8 : mov eax, dword ptr [rsi + 8] ; ret
0x000000000043fbeb : mov eax, ebp ; pop rbp ; ret
0x00000000004220fa : mov eax, ebx ; pop rbx ; ret
0x0000000000495b90 : mov eax, ecx ; pop rbx ; ret
0x0000000000482498 : mov eax, edi ; pop rbx ; ret
0x0000000000437c11 : mov eax, edi ; ret
0x000000000042cfa1 : mov eax, edx ; pop rbx ; ret
0x000000000047d484 : mov eax, edx ; ret
0x000000000043de7e : mov ebp, esi ; jmp rax
0x0000000000499461 : mov ecx, esp ; jmp rax
0x00000000004324fb : mov edi, dword ptr [rbp] ; call rbx
0x0000000000443f34 : mov edi, dword ptr [rdi + 0x30] ; call rax
0x00000000004607e2 : mov edi, dword ptr [rdi] ; call rsi
0x000000000045c71e : mov edi, ebp ; call rax
0x0000000000491e33 : mov edi, ebp ; call rdx
0x00000000004a7a2d : mov edi, ebp ; nop ; call rax
0x000000000045c4c1 : mov edi, ebx ; call rax
```

Generating ROP Chains


```
*** [ Python ] ***

  def create_rop_chain():

    # rop chain generated with mona.py - www.corelan.be
    rop_gadgets = [
      0x10037a10,  # POP EBP # RETN [MSRMfilter03.dll]
      0x10037a10,  # skip 4 bytes [MSRMfilter03.dll]
      0x00000000,  # [-] Unable to find gadget to put 00000201 into ebx
      0x00000000,  # [-] Unable to find gadget to put 00000040 into edx
      0x100204b0,  # POP ECX # RETN [MSRMfilter03.dll]
      0x1006c7b6,  # &Writable location [MSRMfilter03.dll]
      0x1002cdab,  # POP EDI # RETN [MSRMfilter03.dll]
      0x1002a602,  # RETN (ROP NOP) [MSRMfilter03.dll]
      0x1002f4[illegible],  # POP ESI # RETN [MSRMfilter03.dll]
      0x1002ab52,  # JMP [EAX] [MSRMfilter03.dll]
      0x1002ca2d,  # POP EAX # RETN [MSRMfilter03.dll]
      0x00000000,  # [-] Unable to find ptr to &VirtualProtect()
      0x10014720,  # PUSHAD # RETN [MSRMfilter03.dll]
      0x100371f5,  # ptr to 'call esp' [MSRMfilter03.dll]
    ]
    return ''.join(struct.pack('<I', _) for _ in rop_gadgets)

  rop_chain = create_rop_chain()
```

Ropgadet demo