Course material is not allowed during the exam. One A4 sheet (two pages) with handwritten notes allowed.

Try to keep your answers precise and short. You will not get extra points by giving very long answers or by writing down what you know instead of what is asked. Just answer the question that is asked. If your answer does not fit the space provided, it's too long (will not count all points).

Take 10 seconds to think about what you are going to write before writing it.

The points shown in the above table are not final and coefficients can/will be applied to harmonize the grades.

---

Equifax Says Cyberattack May Have Affected 143 Million Customers

By TARA SIEGEL BERHARD, TIFFANY ISU, NICOLE PERKROTH and ROB LIEBER  SEPT. 7, 2017

Hackers stole credit card numbers for 209,000 customers, Equifax said. Kathleen Taylor for The New York Times
1) Check the boxes to say whether the assertions below are valid or not (good answer 1, bad answer -1, result for this question between 0 and 5):

<table>
<thead>
<tr>
<th></th>
<th>True</th>
<th>False</th>
</tr>
</thead>
<tbody>
<tr>
<td>An integer overflow is not necessarily a bug.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Null pointer dereference can be exploited under some conditions.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Recursive function calls is a common cause of Buffer Overflow.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Real world exploits never need to exploit more than one bug/vulnerability.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Information leaks helps to bypass Address Space Layout Randomization.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A buffer overflow is never exploitable if NX and ASLR are present.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

2) **Race conditions**
   a) Explain in a concise way what is a race condition.
   b) Give an example when a race condition can lead to a security vulnerability
3) Stack-based buffer overflow

a) Please explain the vulnerability, please draw a figure to explain it.

b) Provide a piece of (pseudo) code of a vulnerable program.

c) Explain at least 2 different mechanisms that need to be used for exploitation, depending on the countermeasures that are present on the system.

d) Explain 2 Countermeasures against the mechanisms you discussed in “c)”
4) Separation of Privilege

a) Explain in a few words what is separation of privileges.

b) How does it is achieved in an operating system

5) Web Security

a) Please explain what are SQL injection and shell command injection attack. Please highlight what is common and different among those attacks.
6) Equifax breach a deserialization bug known as CVE-2017-9805?
Read the documents related to Equifax and CVE-2017-9805 in appendix.

a) Summarize: what is a deserialization bug?

b) What is the cause of this particular vulnerability (CVE-2017-9805)?

c) Is the opinion (the twitter screenshot) that the breach is due to this CVE vulnerability realistic?
d) Assuming this CVE is the bug that allowed the attackers to breach Equifax webserver, did anything could have been done to prevent this? What?
Kevin Beaumont @GossiTheDog 10h
Equifax's infrastructure is a weird mix of IBM WebSphere, Apache Struts, Java. it's like stepping back in time a decade.

notdan ⭐ @notdan
Cmon Equifax. You have ALL OF OUR INFO.

Replying to @GossiTheDog
CVE-2017-9805 reported July 17, Equifax breach discovered July 29. Wonder how much contemporaneous discovery there was of Struts vuln
5:57 PM - 7 Sep 2017
Severe security vulnerability found in Apache Struts using lgtm.com (CVE-2017-9805)

September 05, 2017 Posted by Bas van Schaik

Originally published on 5 September 15:30 BST. Updated on 6 September: added a warning regarding multiple working exploits having been published by third parties. Included details of Struts version 2.3.34

Security researchers at lgtm.com have discovered a critical remote code execution vulnerability in Apache Struts — a popular open-source framework for developing web applications in the Java programming language. All versions of Struts since 2008 are affected; all web applications using the framework’s popular REST plugin are vulnerable. Shortly after the patched versions of Struts were released on 5 September, multiple working exploits were observed on various internet sites. Users are strongly advised to upgrade their Apache Struts components as a matter of urgency. This vulnerability has been addressed in Struts versions 2.3.34 and 2.5.13.

lgtm provides free software engineering analytics for open-source projects; at the time this post is published, over 50,000 projects are continuously monitored. Anyone can write their own analyses; ranging from checks for enforcing good coding practices to advanced analyses to find security vulnerabilities. The lgtm security team actively helps the open-source community to uncover critical security vulnerabilities in OSS projects.

This particular vulnerability allows a remote attacker to execute arbitrary code on any server running an application built using the Struts framework and the popular REST communication plugin. The weakness is caused by the way Struts deserializes untrusted data. The lgtm security team have a simple working exploit for this vulnerability which will not be published at this stage. At the time of the announcement there is no suggestion that an exploit is publicly available, but it is likely that there will be one soon.

The Apache Struts development team have confirmed the severity of this issue and released a patch today:

This is critical, as all you have to do is use the REST plugin.

The Struts maintainers have posted an announcement on their website and the vulnerability has been assigned CVE 2017-9805. More information about how this vulnerability was found using lgtm.com is available in a separate blog post.

Analyst Fintan Ryan at RedMonk estimates that at least 65% of the Fortune 100 companies are actively using web applications built with the Struts framework. Organizations like Lockheed Martin, the IRS, Citigroup, Vodafone, Virgin Atlantic, Reader’s Digest, Office Depot, and SHOWTIME are known to have developed applications using the framework. This illustrates how widespread the risk is.
When asked for a comment, the Chief Information Security Officer of a Tier 1 bank confirmed that Struts is still used in large numbers of applications and that this finding poses a real threat:

Any security vulnerability can be potentially disastrous, but any that allows Remote Code Execution are especially worrying. This vulnerability is potentially very damaging due to the large number of sites that rely upon this framework. Coupled with the complexities to remediate, as code will have to be changed as opposed to just applying a vendor patch, this has the potential to be worse than the ‘POODLE’ attack was.

Finding this highlights the power that static code analysis can bring, and if something this severe can be in such a well known public library, just imagine what it could find in your code base.

Man Yue Mo, one of the lgtm security researchers who discovered this vulnerability, confirms the criticality:

The Struts framework is used by an incredibly large number and variety of organizations. This vulnerability poses a huge risk, because the framework is typically used for designing publicly-accessible web applications. Struts is used in several airline booking systems as well as a number of financial institutions who use it in internet banking applications. On top of that, it is incredibly easy for an attacker to exploit this weakness: all you need is a web browser. Organizations who use Struts should upgrade their components immediately.

He has written a blog post that describes in more detail how he found this particular vulnerability using the flexible and powerful query language at the heart of lgtm. The lgtm queries flag up software problems and security vulnerabilities on a daily basis. The analysis results for a large number of projects is readily available on lgym.com, including for popular projects like Hadoop, Jetty, Maven, and Storm — all of which have millions of users, and are the building blocks of famous platforms like Twitter, Spotify, Google, and Amazon.

Oege de Moor, CEO and founder of Semmle (the company behind lgtm):

This is as serious as it gets; if remote attackers are allowed to exploit the newly identified vulnerability it can critically damage thousands of enterprises. In the spirit of open source, we want to make sure that the community and industry are aware of these findings as we help uncover critical issues in large numbers of open-source projects. Working with Apache Struts, they were extremely responsive and immediately came up with a clear remediation path.

The technology that powers lgtm is used by many organizations to analyze their software development process and find security vulnerabilities like the one in Struts. These organizations include:
Using QL to find a remote code execution vulnerability in Apache Struts (CVE-2017-9805)

September 05, 2017 Posted by Man Yue Mo

Originally published on 5 September 15:30 BST. Updated on 6 September: added a warning regarding multiple working exploits having been published by third parties. Included details of Struts version 2.3.34

In this post I’ll describe how I customized a standard lgtm query to find a remote code execution vulnerability in Apache Struts. A more general announcement about this vulnerability can be found here. It has been assigned CVE-2017-9805. A release announcement and security bulletin are available on the Apache Struts website. This vulnerability has been addressed in Struts versions 2.3.34 and 2.5.13. Due to the severe nature of this vulnerability, a couple of details (including a working exploit) have been omitted from this post; this information will be added in a few weeks' time.

As of the early morning on 6 September 2017 (GMT), multiple working exploits have been observed on various places on the internet. We strongly advise users of Struts to upgrade to the latest version to mitigate this security risk.

The vulnerability I discovered is a result of unsafe deserialization in Java. Multiple similar vulnerabilities have come to light in recent years, after Chris Frohoff and Gabriel Lawrence discovered a deserialization flaw in Apache Commons Collections that can lead to arbitrary code execution. Many Java applications have since been affected by such vulnerabilities. If you'd like to know more about this type of vulnerability, the lgtm documentation page on this topic is a good place to start.

e)Detecting unsafe deserialization in Struts

lgtm identifies alerts in code using queries written in a specially-designed language: QL. One of the many queries for Java detects potentially unsafe deserialization of user-controlled data. The query identifies situations in which unsanitized data is deserialized into a Java object. This includes data that comes from an HTTP request or from any other socket connection.

This query detects common ways through which user-controlled data flows to a deserialization method. However, some projects use a slightly different approach to receive remote user input. For example, Apache Struts uses the ContentTypeHandler interface. This converts data into Java objects. Since implementations of this interface usually deserialize the data passed to them, every class that implements this interface is potentially of interest. The standard QL query for detecting unsafe deserialization of user-controlled data can easily be adapted to recognize
this additional method for processing user input. This is done by defining a custom data source.
In this case, we are interested in data flowing from the toObject method, which is defined in the ContentTypeHandler interface:

```java
void toObject(Reader in, Object target);
```

The data contained in the first argument in that is passed to toObject should be considered tainted: it is under the control of a remote user and should not be trusted. We want to find places where this tainted data (the source) flows into a deserialization method (a sink) without input validation or sanitization.

The QL DataFlow library provides functionality for tracking tainted data through various steps in the source code. This is known as taint tracking. For example, data gets tracked through various method calls:

```java
IOUtils.copy(remoteUserInput, output);   // output is now also tainted because the function copy preserves the data.
```

To make use of the taint tracking functionality in the DataFlow library, let's define the in argument to ContentTypeHandler.toObject(...) as a tainted source. First, we define how the query should recognize the ContentTypeHandler interface and the method toObject.

```java
/** The ContentTypeHandler Java class in Struts ***/
class ContentTypeHandler extends Interface {
    ContentTypeHandler() {
        this.hasQualifiedName("org.apache.struts2.rest.handler","ContentTypeHandler")
    }
}

/** The method `toObject` */
class ToObjectDeserializer extends Method {
    ToObjectDeserializer() {
        this.getDeclaringType().getASupertype*().instanceof
        ContentTypeHandler and
        this.getSignature = "toObject(java.io.Reader,java.lang.Object)"
    }
}
```
Here we use `getASupertype*()` to restrict the matching to any class that has `ContentTypeHandler` as a supertype.

Next we want to mark the first argument of the `toObject` method as an untrusted data source, and track that data as it flows through the code paths. To do that, we extend the `FlowSource` class in QL's dataflow library:

```java
/** Mark the first argument of `toObject` as a dataflow source **/
class ContentTypeHandlerInput extends FlowSource {
    ContentTypeHandlerInput() {
        exists(ToObjectDeserializer des | des.getParameter(0).getAnAccess() = this )
    }
}
```

Intuitively, this definition says that any access to the first parameter of a `toObject` method, as captured by `ToObjectDeserializer` above, is a flow source. Note that for technical reasons, flow sources have to be expressions. Therefore, we identify all accesses of that parameter (which are expressions) as sources, rather than the parameter itself (which isn't).

Now that we have the definition for a dataflow source, we can look for places where this tainted data is used in an unsafe deserialization method. We don't have to define that method (the sink) ourselves as it is already in the Deserialization of user-controlled data query (line 64: `UnsafeDeserializationSink`, we will need to copy its definition into the query console). Using this, our final query becomes:

```sql
from ContentTypeHandlerInput source, UnsafeDeserializationSink sink
where source.flowsTo(sink)
select source, sink
```

Here we use the `.flowsTo` predicate in `FlowSource` for tracking so that we only identify the cases when unsafe deserialization is performed on a `ContentTypeHandlerInput` source.

When I ran the customized query on Struts there was exactly one result (Running it now will yield no result as the fix has been applied). I verified that it was a genuine remote code execution vulnerability before reporting it to the Struts security team. They have been very quick and responsive in working out a solution even though it is a fairly non-trivial task that requires API changes. Due to the severity of this finding I will not...
disclose more details at this stage. Rather, I will update this blog post in a couple of weeks' time with more information.

f) Vendor Response
   • 17 July 2017: Initial disclosure.
   • 02 August 2017: API changes in preparation for patch.
   • 14 August 2017: Patch from Struts for review.
   • 16 August 2017: Vulnerability officially recognized as CVE-2017-9805
   • 5 September 2017: Struts version 2.5.13 released

Mitigate unsafe deserialization risk with lgtm

lgtm runs the standard Deserialization of user-controlled data query on all Java projects. If your project uses deserialization frameworks detected by that query, and has user-controlled data reaching a deserialization method, you may see relevant alerts for this query on lgtm.com. Check any results carefully. You can also enable lgtm’s pull request integration to prevent serious security issues like these from being merged into the code base in the first place.

If your project uses other deserialization frameworks, then you can use the query console to create your own custom version of the standard query.
Equifax Says Cyberattack May Have Affected 143 Million Customers

By TARA SIEGEL BERNARD, TIFFANY HSU, NICOLE PERLROTH and RON LIEBER  SEPT. 7, 2017

Equifax, one of the three major consumer credit reporting agencies, said on Thursday that hackers had gained access to company data that potentially compromised sensitive information for 143 million American consumers, including Social Security numbers and driver’s license numbers.

The attack on the company represents one of the largest risks to personally sensitive information in recent years, and is the third major cybersecurity threat for the agency since 2015.

Equifax, based in Atlanta, is a particularly tempting target for hackers. If identity thieves wanted to hit one place to grab all the data needed to do the most damage, they would go straight to one of the three major credit reporting agencies.

“This is about as bad as it gets,” said Pamela Dixon, executive director of the World Privacy Forum, a nonprofit research group. “If you have a credit report, chances are you may be in this breach. The chances are much better than 50 percent.”

Criminals gained access to certain files in the company’s system from mid-May to July by exploiting a weak point in website software, according to an investigation by Equifax and security consultants. The company said that it
discovered the intrusion on July 29 and has since found no evidence of unauthorized activity on its main consumer or commercial credit reporting databases.

In addition to the other material, hackers were also able to retrieve names, birth dates and addresses. Credit card numbers for 209,000 consumers were stolen, while documents with personal information used in disputes for 182,000 people were also taken.

Other cyberattacks, such as the two breaches that Yahoo announced in 2016, have eclipsed the penetration at Equifax in sheer size, but the Equifax attack is worse in terms of severity. Thieves were able to siphon far more personal information — the keys that unlock consumers’ medical histories, bank accounts and employee accounts.

"On a scale of 1 to 10 in terms of risk to consumers, this is a 10," said Avivah Litan, a fraud analyst at Gartner.

An F.B.I. spokesperson said the agency was aware of the breach and was tracking the situation.

Last year, identity thieves successfully made off with critical W-2 tax and salary data from an Equifax website. And earlier this year, thieves again stole W-2 tax data from an Equifax subsidiary, TALX, which provides online payroll, tax and human resources services to some of the nation’s largest corporations.

Cybersecurity professionals criticized Equifax on Thursday for not improving its security practices after those previous thefts, and they noted that thieves were able to get the company’s crown jewels through a simple website vulnerability.

"Equifax should have multiple layers of controls" so if hackers manage to break in, they can at least be stopped before they do too much damage, Ms. Litan said.

Potentially adding to criticism of the company, three senior executives, including the company's chief financial officer, John Gamble, sold shares
worth almost $1.8 million in the days after the breach was discovered. The shares were not part of a sale planned in advance, Bloomberg reported.

The company handles data on more than 820 million consumers and more than 91 million businesses worldwide and manages a database with employee information from more than 7,100 employers, according to its website.

Equifax also houses much of the data that is supposed to be a backstop against security breaches. The agency offers a service that provides companies with the questions and answers needed for their account recovery, in the event customers lose access to their accounts.

“If that information is breached, you’ve lost that backstop,” said Patrick Harding, the chief technology officer at Ping Identity, a Denver-based identity management company.

Equifax said that, in addition to reporting the breach to law enforcement, it had hired a cybersecurity firm to conduct a review to determine the scale of the invasion. The investigation is expected to wrap up in the next few weeks.

“This is clearly a disappointing event for our company, and one that strikes at the heart of who we are and what we do,” Richard F. Smith, chairman and chief executive of Equifax, said in a statement. “Confronting cybersecurity risks is a daily fight.”

Using the data stolen from Equifax, identity thieves can impersonate people with lenders, creditors and service providers, who rely on personal identity information from Equifax to make financial decisions regarding potential customers.

Equifax has created a website, www.equifaxsecurity2017.com, to help consumers determine whether their data was at risk.

People can go to the Equifax website to see if their information has been compromised. The site encourages customers to offer their last name and the last six digits of their Social Security number. When they do, however, they do not necessarily get confirmation about whether they were affected.
Instead, the site provides an enrollment date for its protection service, and it may not start for several days.

The company also suggests getting a free copy of your credit report from the three major credit bureaus: Equifax, Experian and TransUnion. These are available at annualcreditreport.com. It also suggests contacting a law enforcement agency if you believe any stolen information has already been used in some way.

Equifax’s credit protection service, which is free for one year for consumers who enroll by Nov. 21, is available to everyone and not just the victims of the breach.

Equifax is offering consumers the ability to freeze their Equifax credit reports, said John Ulzheimer, a consumer credit expert who often does expert witness work for banks and credit unions and worked at Equifax in the 1990s. Thieves could have information stolen from Equifax and used it to open accounts with creditors that use Experian or TransUnion.

“It’s like locking one of three doors in your house and leaving the other two unlocked,” Mr. Ulzheimer said. “You’re hoping the thief stumbles on the locked door.” He recommended that all those affected immediately place a fraud alert on all three of their credit files, which anyone can do for free.

Equifax’s offer of one year of free protection falls short of what consumers really need, because their information can be bought and sold by hackers for years to come, Mr. Ulzheimer added.

Beyond compromising the personal data of millions of consumers, the breach also poses a potential national security threat. In recent years, Chinese nation-state hackers have breached insurers like Anthem and federal agencies, siphoning detailed personal and medical information. These hackers go wide in their assaults in an effort to build databases of Americans’ personal information, which can be used for blackmail or future attacks.

Governments regularly buy stolen personal information on the so-called Dark Web, security experts say. The black market sites where this information is sold are far more exclusive than black markets where stolen
credit card data is sold. Interested buyers are even asked to submit to background checks before they are admitted.

“Cyberwar is in large part conducted through data mining and cyberintelligence,” Ms. Litan said. “This is also a Homeland Security risk as enemy nation states build databases of Americans that they then use to get to their targets, for example a network operator at a power grid, or a defense contractor at a missile defense company.”

Sen. Mark R. Warner, a Virginia Democrat who co-founded the Senate Cybersecurity Caucus, said he believed the severity of the Equifax breach raised serious questions about whether Congress needed to rethink data protection policies.

“It is no exaggeration to suggest that a breach such as this — exposing highly sensitive personal and financial information central for identity management and access to credit — represents a real threat to the economic security of Americans,” he said in a statement.

A version of this article appears in print on September 8, 2017, on Page A1 of the New York edition with the headline: Equifax Attack Exposes Data Of 143 Million.

© 2017 The New York Times Company
Cybersecurity Incident & Important Consumer Information

- Home
- Consumer Notice
- FAQs
- Potential Impact
- Enroll
- TrustedID Premier
- Contact Us

Equifax Announces Cybersecurity Incident Involving Consumer Information

Rick Smith, Chairman and CEO of Equifax, on Cybersecurity Incident Involving Con...

No Evidence of Unauthorized Access to Core Consumer or Commercial Credit Reporting Databases

Company to Offer Free Identity Theft Protection and Credit File Monitoring to All U.S. Consumers

September 7, 2017 — Equifax Inc. (NYSE: EFX) today announced a cybersecurity incident potentially impacting approximately 143 million U.S. consumers. Criminals exploited a U.S. website application vulnerability to gain access to certain files. Based on the company’s investigation, the unauthorized access
occurred from mid-May through July 2017. The company has found no evidence of unauthorized activity on Equifax’s core consumer or commercial credit reporting databases.

The information accessed primarily includes names, Social Security numbers, birth dates, addresses and, in some instances, driver’s license numbers. In addition, credit card numbers for approximately 209,000 U.S. consumers, and certain dispute documents with personal identifying information for approximately 182,000 U.S. consumers, were accessed. As part of its investigation of this application vulnerability, Equifax also identified unauthorized access to limited personal information for certain UK and Canadian residents. Equifax will work with UK and Canadian regulators to determine appropriate next steps. The company has found no evidence that personal information of consumers in any other country has been impacted.

Equifax discovered the unauthorized access on July 29 of this year and acted immediately to stop the intrusion. The company promptly engaged a leading, independent cybersecurity firm that has been conducting a comprehensive forensic review to determine the scope of the intrusion, including the specific data impacted. Equifax also reported the criminal access to law enforcement and continues to work with authorities. While the company’s investigation is substantially complete, it remains ongoing and is expected to be completed in the coming weeks.

“This is clearly a disappointing event for our company, and one that strikes at the heart of who we are and what we do. I apologize to consumers and our business customers for the concern and frustration this causes,” said Chairman and Chief Executive Officer, Richard F. Smith. “We pride ourselves on being a leader in managing and protecting data, and we are conducting a thorough review of our overall security operations. We also are focused on consumer protection and have developed a comprehensive portfolio of services to support all U.S. consumers, regardless of whether they were impacted by this incident.”

Equifax has established a dedicated website, www.equifaxsecurity2017.com, to help consumers determine if their information has been potentially impacted and to sign up for credit file monitoring and identity theft protection. The offering, called TrustedID Premier, includes 3-Bureau credit monitoring of Equifax, Experian and TransUnion credit reports; copies of Equifax credit reports; the ability to lock and unlock Equifax credit reports; identity theft insurance; and Internet scanning for Social Security numbers – all complimentary to U.S. consumers for one year. The website also provides additional information on steps consumers can take to protect their personal information. Equifax recommends that consumers with additional questions visit www.equifaxsecurity2017.com or contact a dedicated call center at 866-447-7559, which the company set up to assist consumers. The call center is open every day (including weekends) from 7:00 a.m. – 1:00 a.m. Eastern time.

In addition to the website, Equifax will send direct mail notices to consumers whose credit card numbers or dispute documents with personal identifying information were impacted. Equifax also is in the process of contacting U.S. state and federal regulators and has sent written notifications to all U.S. state attorneys general, which includes Equifax contact information for regulator inquiries.

Equifax has engaged a leading, independent cybersecurity firm to conduct an assessment and provide recommendations on steps that can be taken to help prevent this type of incident from happening again.

CEO Smith said, “I’ve told our entire team that our goal can’t be simply to fix the problem and move on. Confronting cybersecurity risks is a daily fight. While we’ve made significant investments in data security, we recognize we must do more. And we will.”

About Equifax

Equifax is a global information solutions company that uses trusted unique data, innovative analytics, technology and industry expertise to power organizations and individuals around the world by transforming knowledge into insights that help make more informed business and personal decisions.

Headquartered in Atlanta, Ga., Equifax operates or has investments in 24 countries in North America, Central and South America, Europe and the Asia Pacific region. It is a member of Standard & Poor’s (S&P) 500® Index, and its common stock is traded on the New York Stock Exchange (NYSE) under the symbol EFX. Equifax employs approximately 9,900 employees worldwide.
Forward-Looking Statements

This release contains forward-looking statements and forward-looking information. These statements can be identified by expressions of belief, expectation or intention, as well as estimates and statements that are not historical fact. These statements are based on certain factors and assumptions with respect to the investigation of the cybersecurity incident to date. While the company believes these factors and assumptions to be reasonable based on information currently available, they may prove to be incorrect.

Several factors could cause actual results to differ materially from those expressed or implied in the forward-looking statements, including, but not limited to, the final results of the investigation, including the final scope of the intrusion, the type of information accessed and the number of consumers impacted. A summary of additional risks and uncertainties can be found in our Annual Report on Form 10-K for the year ended December 31, 2016, including without limitation under the captions “Item 1. Business — Governmental Regulation” and “— Forward-Looking Statements” and “Item 1A. Risk Factors,” and in our other filings with the U.S. Securities and Exchange Commission. Forward-looking statements are given only as at the date of this release and the company disclaims any obligation to update or revise the forward-looking statements, whether as a result of new information, future events or otherwise, except as required by law.

Contacts:

Ines Gutzmer
Corporate Communications
mediainquiries@equifax.com
404-885-8555

Copyright 2017 Equifax, Inc. All rights reserved

Equifax and the Equifax marks used herein are trademarks of Equifax Inc. Other product and company names mentioned herein are the property of their respective owners.
Description

Description Summary
The application deserializes untrusted data without sufficiently verifying that the resulting data will be valid.

Extended Description
It is often convenient to serialize objects for communication or to save them for later use. However, deserialized data or code can often be modified without using the provided accessor functions if it does not use cryptography to protect itself. Furthermore, any cryptography would still be client-side security -- which is a dangerous security assumption.

Data that is untrusted can not be trusted to be well-formed.

When developers place no restrictions on "gadget chains," or series of instances and method invocations that can self-execute during the deserialization process (i.e., before the object is returned to the caller), it is sometimes possible for attackers to leverage them to perform unauthorized actions, like generating a shell.

Applicable Platforms

Languages
Java
Ruby
PHP
Python
Node.js
Language-independent

Common Consequences

<table>
<thead>
<tr>
<th>Scope</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integrity</td>
<td><strong>Technical Impact:</strong> <em>Modify application data; Unexpected state</em></td>
</tr>
<tr>
<td>Availability</td>
<td><strong>Technical Impact:</strong> <em>DoS: resource consumption (CPU)</em></td>
</tr>
<tr>
<td>Authorization</td>
<td><strong>Technical Impact:</strong> <em>Other</em></td>
</tr>
<tr>
<td>Scope</td>
<td>Effect</td>
</tr>
<tr>
<td>Technical Impact: <em>Varies by context</em></td>
<td></td>
</tr>
<tr>
<td>The consequences can vary widely, because it depends on which objects or methods are being deserialized, and how they are used.</td>
<td></td>
</tr>
</tbody>
</table>
Code could potentially make the assumption that information in the deserialized object is valid. Functions that make this dangerous assumption could be exploited.

**Likelihood of Exploit**
Medium

**Demonstrative Examples**

**Example 1**
This code snippet deserializes an object from a file and uses it as a UI button:

This code does not attempt to verify the source or contents of the file before deserializing it. An attacker may be able to replace the intended file with a file that contains arbitrary malicious code which will be executed when the button is pressed.

**Example 2**
In Python, the Pickle library handles the serialization and deserialization processes. In this example derived from [R.502.7], the code receives and parses data, and afterwards tries to authenticate a user based on validating a token.

Unfortunately, the code does not verify that the incoming data is legitimate. An attacker can construct a illegitimate, serialized object "AuthToken" that instantiates one of Python's subprocesses to execute arbitrary commands. For instance, the attacker could construct a pickle that leverages Python's subprocess module, which spawns new processes and includes a number of arguments for various uses. Since Pickle allows objects to define the process for how they should be unpickled, the attacker can direct the unpickle process to call Popen in the subprocess module and execute /bin/sh.

**Potential Mitigations**

**Phases: Architecture and Design; Implementation**
If available, use the signing/sealing features of the programming language to assure that deserialized data has not been tainted. For example, a hash-based message authentication code (HMAC) could be used to ensure that data has not been modified.

**Phase: Implementation**
When deserializing data, populate a new object rather than just deserializing. The result is that the data flows through safe input validation and that the functions are safe.

**Phase: Implementation**
Explicitly define final readObject() to prevent deserialization. An example of this is:

**Phases: Architecture and Design; Implementation**
Make fields transient to protect them from deserialization.
An attempt to serialize and then deserialize a class containing transient fields will result in NULLs where the transient data should be. This is an excellent way to prevent time, environment-based, or sensitive variables from being carried over and used improperly.

**Phase: Implementation**
Avoid having unnecessary types or gadgets available that can be leveraged for malicious ends. This limits the potential for unintended or unauthorized types and gadgets to be leveraged by the attacker. Whitelist acceptable classes. Note: new gadgets are constantly being discovered, so this alone is not a sufficient mitigation.

<table>
<thead>
<tr>
<th>Nature</th>
<th>Type</th>
<th>ID</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>858</td>
<td>CERT Java Secure Coding Section 13 - Serialization (SER)</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>913</td>
<td>Improper Control of Dynamically-Managed Code Resources</td>
</tr>
<tr>
<td>ChildOf</td>
<td>C</td>
<td>994</td>
<td>SFP Secondary Cluster: Tainted Input to Variable</td>
</tr>
<tr>
<td>PeerOf</td>
<td>B</td>
<td>915</td>
<td>Improperly Controlled Modification of Dynamically-Determined Object Attributes</td>
</tr>
<tr>
<td>MemberOf</td>
<td>V</td>
<td>884</td>
<td>CWE Cross-section</td>
</tr>
</tbody>
</table>
Introduction

This article is focused on providing clear, actionable guidance for safely deserializing untrusted data in your applications.

What is Deserialization?

Serialization is the process of turning some object into a data format that can be restored later. People often serialize objects in order to save them to storage, or to send as part of communications. Deserialization is the reverse of that process -- taking data structured from some format, and rebuilding it into an object. Today, the most popular data format for serializing data is JSON. Before that, it was XML.

However, many programming languages offer a native capability for serializing objects. These native formats usually offer more features than JSON or XML, including customizability of the serialization process. Unfortunately, the features of these native deserialization mechanisms can be repurposed for malicious effect when operating on untrusted data. Attacks against deserializers have been found to allow denial-of-service, access control, and remote code execution attacks.

Guidance on Deserializing Objects Safely

1. Introduction
2. What is Deserialization?
3. Guidance on Deserializing Objects Safely
   - 3.1 Java
     - 3.1.1 Prevent Data Leakage and Trusted Field Clobbering
     - 3.1.2 Prevent Deserialization of Domain Objects
     - 3.1.3 Harden Your Own java.io.ObjectInputStream
     - 3.1.4 Harden All java.io.ObjectInputStream Usage with an Agent
   - 4 Language-Agnostic Methods for Deserializing Safely
     - 4.1 Using Alternative Data Formats
     - 4.2 Only Deserialize Signed Data
   - 5 References
   - 6 Authors and Primary Editors
     - 6.1 Other Cheatsheets
The following language-specific guidance attempts to enumerate safe methodologies for deserializing data that can't be trusted.

**Java**

The following techniques are all good for preventing attacks against deserialization against Java's Serializable format (http://docs.oracle.com/javase/7/docs/api/java/io/Serializable.html).

Implementation: In your code, override the ObjectInputStream#resolveClass() method to prevent arbitrary classes from being deserialized. This safe behavior can be wrapped in a library like SerialKiller. Implementation: Use a safe replacement for the generic readObject() method as seen here. Note that this addresses "billion laughs" type attacks by checking input length and number of objects deserialized.

**Prevent Data Leakage and Trusted Field Clobbering**

If there are members of the object graph that should never be controlled by end users during deserialization or exposed to users during serialization, they should be marked with the transient keyword (https://docs.oracle.com/javase/7/docs/platform/serialization/spec/serial-arch.html#6250).

**Prevent Deserialization of Domain Objects**

Some of your application objects may be forced to implement Serializable due to their hierarchy. To guarantee that your application objects can't be deserialized, a readObject() should be declared (with a final modifier) which always throws an exception.

```java
private final void readObject(ObjectInputStream in) throws java.io.IOException {
    throw new java.io.IOException("Cannot be deserialized");
}
```

**Harden Your Own java.io.ObjectInputStream**

The java.io.ObjectInputStream class is used to deserialize objects. It's possible to harden its behavior by subclassing it. This is the best solution if:

- You can change the code that does the deserialization
- You know what classes you expect to deserialize

The general idea is to override ObjectInputStream.html#resolveClass() (http://docs.oracle.com/javase/7/docs/api/java/io/ObjectInputStream.html#resolveClass(java.io.ObjectStreamClass)) in order to restrict which classes are allowed to be deserialized. Because this call happens before a readObject() is called, you can be sure that no deserialization activity will occur unless the type is one that you wish to allow. A simple example of this shown here, where the the LookAheadObjectInputStream class is guaranteed not to deserialize any other type besides the Bicycle class:

```java
public class LookAheadObjectInputStream extends ObjectInputStream {
    public LookAheadObjectInputStream(InputStream inputStream) throws IOException {
        super(inputStream);
    }

    /**
     * Only deserialize instances of our expected Bicycle class
     */
    @Override
    protected Class<?> resolveClass(ObjectStreamClass desc) throws IOException,
    ClassNotFoundException {
        if (!desc.getName().equals(Bicycle.class.getName())) {
            throw new InvalidClassException("Unauthorized deserialization attempt",
                    desc.getName());
        }
        return super.resolveClass(desc);
    }
}
```
More complete implementations of this approach have been proposed by various community members:

- NibbleSec (https://github.com/ikkisoft/SerialKiller) - a library that allows whitelisting and blacklisting of classes that are allowed to be deserialized
- Contrast Security (http://www.contrastsecurity.com/security-influencers/java-serialization-vulnerability-threatens-millions-of-applications) - a utility method that allows whitelisting of classes to deserialize, as well as other thresholds.
- IBM (https://www.ibm.com/developerworks/library/se-lookahead/) - the seminal protection, written years before the most devastating exploitation scenarios were envisioned.

**Harden All java.io.ObjectInputStream Usage with an Agent**

As mentioned above, the java.io.ObjectInputStream class is used to deserialize objects. It's possible to harden its behavior by subclassing it. However, if you don't own the code or can't wait for a patch, using an agent to weave in hardening to java.io.ObjectInputStream is the best solution.

Globally changing ObjectInputStream is only safe for blacklisting known malicious types, because it's not possible to know for all applications what the expected classes to be deserialized are. Fortunately, there are very few classes needed in the blacklist to be safe from all the known attack vectors, today. It's inevitable that more "gadget" classes will be discovered that can be abused. However, there is an incredible amount of vulnerable software exposed today, in need of a fix. In some cases, "fixing" the vulnerability may involve re-architecting messaging systems and breaking backwards compatibility as developers move towards not accepting serialized objects.

To enable these agents, simply add a new JVM parameter:

```
-javaagent:name-of-agent.jar
```

Agents taking this approach have been released by various community members:

- Invoker Defender by Go-CD (https://github.com/gocd/invoker-defender)
- rO0 by Contrast Security (https://github.com/Contrast-Security-OSS/contrast-rO0)
- Contrast Enterprise by Contrast Security (commercial product) (https://www.contrastsecurity.com)

A similar, but less scalable approach would be to manually patch and bootstrap your JVM's ObjectInputStream. Guidance on this approach is available here (https://github.com/wsargent/paranoid-java-serialization).

**Language-Agnostic Methods for Deserializing Safely**

**Using Alternative Data Formats**

A great reduction of risk is achieved by avoiding native deserialization formats. By switching to a pure data format like JSON or XML, you lessen the chance of custom deserialization logic being repurposed towards malicious ends.

Many applications rely on a data-transfer object pattern (https://en.wikipedia.org/wiki/Data_transfer_object) that involves creating a separate domain of objects for the explicit purpose data transfer. Of course, it's still possible that the application will make security mistakes after a pure data object is parsed.

**Only Deserialize Signed Data**

If the application knows before deserialization which messages will need to be processed, they could sign them as part of the serialization process. The application could then choose not to deserialize any message which didn't have an authenticated signature.
References

- Deserialization of untrusted data
- Java Deserialization Attacks - German OWASP Day 2016

Authors and Primary Editors

Arshan Dabirsiaghi - arshan [at] contrastsecurity dot org

Other Cheatsheets

<table>
<thead>
<tr>
<th>Developer / Builder</th>
<th>Cheat Sheets</th>
</tr>
</thead>
<tbody>
<tr>
<td>3rd Party Javascript Management</td>
<td>Access Control</td>
</tr>
<tr>
<td>· AJAX Security Cheat Sheet</td>
<td>· Authentication (ES)</td>
</tr>
<tr>
<td>· Bean Validation Cheat Sheet</td>
<td>· Choosing and Using Security Questions</td>
</tr>
<tr>
<td>· Clickjacking Defense</td>
<td>· C-Based Toolchain Hardening</td>
</tr>
<tr>
<td>· Credential Stuffing Prevention Cheat Sheet</td>
<td>· Cross-Site Request Forgery (CSRF) Prevention</td>
</tr>
<tr>
<td>· Cryptographic Storage</td>
<td>· Deserialization</td>
</tr>
<tr>
<td>· DOM based XSS Prevention</td>
<td>· Forgot Password</td>
</tr>
<tr>
<td>· HTML5 Security</td>
<td>· HTTP Strict Transport Security</td>
</tr>
<tr>
<td>· Injection Prevention Cheat Sheet</td>
<td>· Injection Prevention Cheat Sheet in Java</td>
</tr>
<tr>
<td>· Input Validation</td>
<td>· JSON Web Token (JWT) Cheat Sheet for Java</td>
</tr>
<tr>
<td>· JAAS</td>
<td>· Input Validation</td>
</tr>
<tr>
<td>· LDAP Injection Prevention</td>
<td>· Java Library Injection Cheat Sheet for Java</td>
</tr>
<tr>
<td>· Logging</td>
<td>· Java Library Injection Prevention for Java Libraries</td>
</tr>
<tr>
<td>· Mass Assignment Cheat Sheet</td>
<td>· .NET Security</td>
</tr>
<tr>
<td>· .NET Security</td>
<td>· OWASP Top Ten</td>
</tr>
<tr>
<td>· Password Storage</td>
<td>· Pinning</td>
</tr>
<tr>
<td>· Pinning</td>
<td>· Query Parameterization</td>
</tr>
<tr>
<td>· Ruby on Rails</td>
<td>· Session Management</td>
</tr>
<tr>
<td>· SAML Security</td>
<td>· SQL Injection Prevention</td>
</tr>
<tr>
<td>· SQL Injection Prevention</td>
<td>· Transaction Authorization</td>
</tr>
<tr>
<td>· Transaction Authorization</td>
<td>· Transport Layer Protection</td>
</tr>
<tr>
<td>· Unvalidated Redirects and Forwards</td>
<td>· Unvalidated Redirects and Forwards</td>
</tr>
<tr>
<td>· User Privacy Protection</td>
<td>· Web Service Security</td>
</tr>
<tr>
<td>· Web Service Security</td>
<td>· XSS (Cross Site Scripting) Prevention</td>
</tr>
<tr>
<td>· XSS Filter Evasion</td>
<td>· XML External Entity (XXE) Prevention Cheat Sheet</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Assessment / Breaker</th>
<th>Cheat Sheets</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attack Surface Analysis</td>
<td>· REST Assessment</td>
</tr>
<tr>
<td>· Web Application Security Testing</td>
<td>· XML Security Cheat Sheet</td>
</tr>
<tr>
<td>· XML Security Cheat Sheet</td>
<td>· XSS Filter Evasion</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Mobile</th>
<th>Cheat Sheets</th>
</tr>
</thead>
<tbody>
<tr>
<td>Android Testing</td>
<td>IOS Developer</td>
</tr>
<tr>
<td>· Mobile Jailbreaking</td>
<td>Virtual Patching</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>OpSec / Defender</th>
<th>Cheat Sheets</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application Security Architecture</td>
<td>Business Logic Security</td>
</tr>
<tr>
<td>· Command Injection Defense Cheat Sheet</td>
<td>· Command Injection Defense Cheat Sheet</td>
</tr>
<tr>
<td>· Content Security Policy</td>
<td>· Denial of Service Cheat Sheet</td>
</tr>
<tr>
<td>· Grails Secure Code Review</td>
<td>· Insecure Direct Object Reference Prevention</td>
</tr>
<tr>
<td>· IOS Application Security Testing</td>
<td>· Key Management</td>
</tr>
<tr>
<td>· Key Management</td>
<td>· PHP Security</td>
</tr>
<tr>
<td>· REST Security</td>
<td>· Regular Expression Security Cheatsheet</td>
</tr>
<tr>
<td>· Regular Expression Security Cheatsheet</td>
<td>· Secure Coding</td>
</tr>
<tr>
<td>· Secure Coding</td>
<td>· Secure SDLC</td>
</tr>
<tr>
<td>· Secure SDLC</td>
<td>· Threat Modeling</td>
</tr>
<tr>
<td>· Threat Modeling</td>
<td>· Vulnerability Disclosure</td>
</tr>
</tbody>
</table>

All Pages In This Category


Category: Cheatsheets

- This page was last modified on 8 June 2017, at 15:40.
- Content is available under Creative Commons Attribution-ShareAlike unless otherwise noted.
### CVE-ID

**CVE-2017-9805**

<table>
<thead>
<tr>
<th>Learn more at National Vulnerability Database (NVD)</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Severity Rating • Fix Information • Vulnerable Software Versions • SCAP Mappings</td>
</tr>
</tbody>
</table>

### Description

**RESERVED** This candidate has been reserved by an organization or individual that will use it when announcing a new security problem. When the candidate has been publicized, the details for this candidate will be provided.

### References

**Note:** References are provided for the convenience of the reader to help distinguish between vulnerabilities. The list is not intended to be complete.

### Assigning CNA

N/A

### Date Entry Created

20170621

**Disclaimer:** The entry creation date may reflect when the CVE-ID was allocated or reserved, and does not necessarily indicate when this vulnerability was discovered, shared with the affected vendor, publicly disclosed, or updated in CVE.
<table>
<thead>
<tr>
<th>Proposed (Legacy)</th>
</tr>
</thead>
<tbody>
<tr>
<td>N/A</td>
</tr>
</tbody>
</table>

This is an entry on the [CVE list](https://cve.mitre.org), which standardizes names for security problems.

**SEARCH CVE USING KEYWORDS:**

You can also search by reference using the [CVE Reference Maps](https://cve.mitre.org).

**For More Information:** cve@mitre.org
Summary

Possible Remote Code Execution attack when using the Struts REST plugin with XStream handler to handle XML payloads

<table>
<thead>
<tr>
<th>Who should read this</th>
<th>All Struts 2 developers and users</th>
</tr>
</thead>
<tbody>
<tr>
<td>Impact of vulnerability</td>
<td>A RCE attack is possible when using the Struts REST plugin with XStream handler to deserialise XML requests</td>
</tr>
<tr>
<td>Maximum security rating</td>
<td>Critical</td>
</tr>
<tr>
<td>Recommendation</td>
<td>Upgrade to Struts 2.5.13 or Struts 2.3.34</td>
</tr>
<tr>
<td>Affected Software</td>
<td>Struts 2.1.2 - Struts 2.3.33, Struts 2.5 - Struts 2.5.12</td>
</tr>
<tr>
<td>Reporter</td>
<td>Man Yue Mo &lt;mmo at semmle dot com&gt; (lgtm.com / Semmle). More information on the lgtm.com blog: <a href="https://lgtm.com/blog">https://lgtm.com/blog</a></td>
</tr>
<tr>
<td>CVE Identifier</td>
<td>CVE-2017-9805</td>
</tr>
</tbody>
</table>

Problem

The REST Plugin is using a XStreamHandler with an instance of XStream for deserialization without any type filtering and this can lead to Remote Code Execution when deserializing XML payloads.

Solution

Upgrade to Apache Struts version 2.5.13 or 2.3.34.

Backward compatibility

It is possible that some REST actions stop working because of applied default restrictions on available classes. In such case please investigate the new interfaces that was introduced to allow define class restrictions per action, those interfaces are:

- org.apache.struts2.rest.handler.AllowedClasses
- org.apache.struts2.rest.handler.AllowedClassNames
- org.apache.struts2.rest.handler.XStreamPermissionProvider

Workaround
The best option is to remove the Struts REST plugin when not used. Alternatively you can only upgrade the plugin by dropping in all the required JARs (plugin plus all dependencies). Another options is to limit th plugin to server normal pages and JSONs only:

1. Disable handling XML pages and requests to such pages

```xml
<constant name="struts.action.extension" value="xhtml,,json" />
```

2. Override getContentType in XStreamHandler

```java
public class MyXStreamHandler extends XStreamHandler {
    public String getContentType() {
        return "not-existing-content-type-@;/&%$#@";
    }
}
```

3. Register the handler by overriding the one provided by the framework in your struts.xml

```xml
<bean type="org.apache.struts2.rest.handler.ContentTypeHandler" name="myXStreamHandmer" class="">
    <constant name="struts.rest.handlerOverride.xml" value="myXStreamHandler"/>
</bean>
```